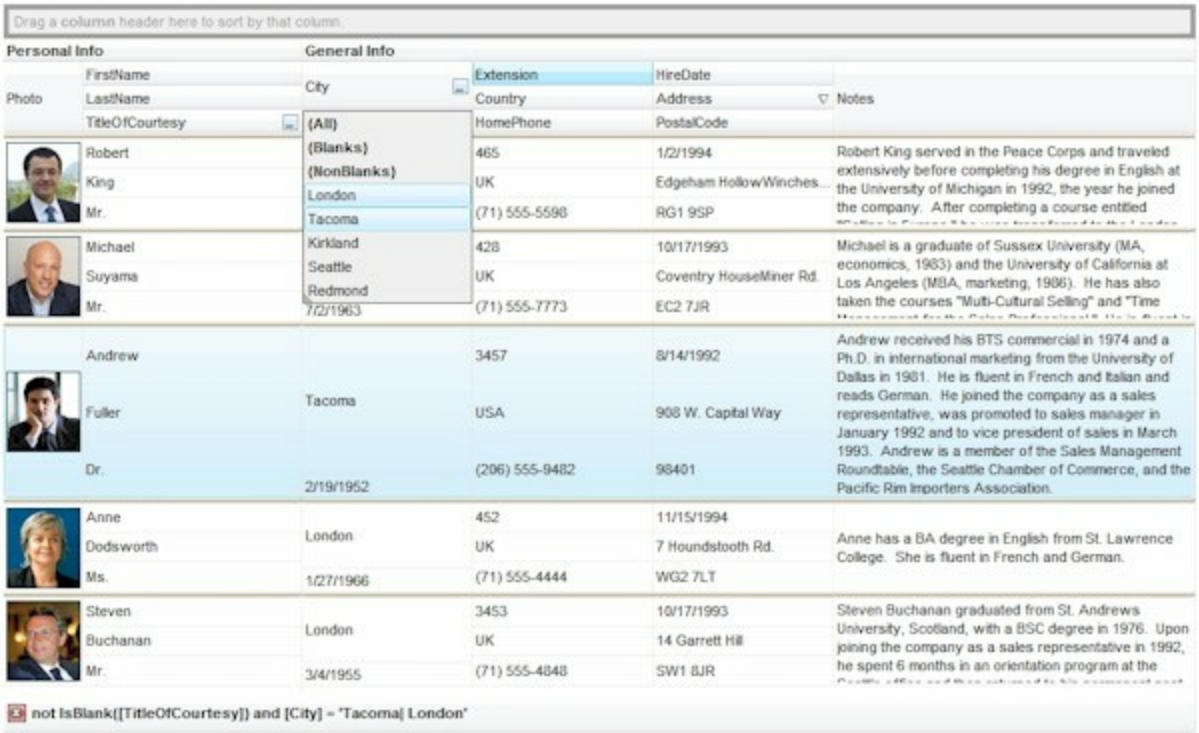
ExGrid

Exontrol's new exGrid control an easy-to-implement grid control, provides swift and robust performance and a wide range of formatting features that distinguish it from other grids. The ExGrid is a multi-purpose data visualization system that can display information as a tree, a grid or list, or a combination of both - in either data-bound or unbound mode. This unique synergy between a traditional grid and a traditional tree view allows you to create cutting-edge and visually appealing application interfaces for your end-users.

Features include:

- Ability to display the data hierarchical (**tree**) , tabular (**list**) or as **card** view
- **Print** and Print Preview support.
- **WYSWYG Template/Layout Editor** support (Ability to load and save control's content to template format).
- **Skinnable Interface** support (ability to apply a skin to the any background part)
- **Custom Row Designer** (Have your rows display however you want with the control row layout capabilities)
- **ADO** and **DAO** support
- Undo/Redo support
- It includes editors like: mask, date, drop down list, check box list, memo fields, spin, slider, OLE Object viewer, color, buttons and more.
- Ability to use custom **ActiveX** control as built-in **editors**.
- **ActiveX hosting** (you can place any ActiveX component as a child item of the control).
- **/NET Control hosting** (you can place any /NET control as a child item of the control).
- **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.
- Ability to show the control's element from right-to-left for Hebrew, Arabic and other **RTL** languages.
- **Conditional Format** support
- **Computed Fields** support
- **Total Fields** support (Aggregate functions: sum, min, max, count, avg)
- Owner Draw Support.
- **Sorting by Single or Multiple Column** support
- Ability to specify **unsortable** items.
- **Single/Multiple Lines** or **Multiple Levels Header** support

- Multiple selection
- **Incremental search**
- Mouse wheel support
- Ability to load icons and pictures from BASE64 encoded strings.
- Background picture support.
- Ability to handle more than 2,000,000,000 records, using the **virtual mode.**
- Ability to get data using I**Unbound**Handler notification interface
- Support for dragging, sorting or resizing columns as well
- Any cell supports **Built-in HTML format.**
- **"Split Cells"** support.
- **"Merge Cells"** support.
- **Locked/fixed rows/items** support.
- **Cell's built-in editor** support.
- **Not selectable** items support.
- Ability to group data using **divider items**. Allows you to merge cells as well.
- Support for drag and drop the items
- Ability to apply attributes like: font, color, icon, picture, radio buttons, check boxes to any cell
- Ability to assign multiple icons to a cell.
- Item can have different height, multi-line items
- Cells can have HTML multiple lines tooltips
- 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.

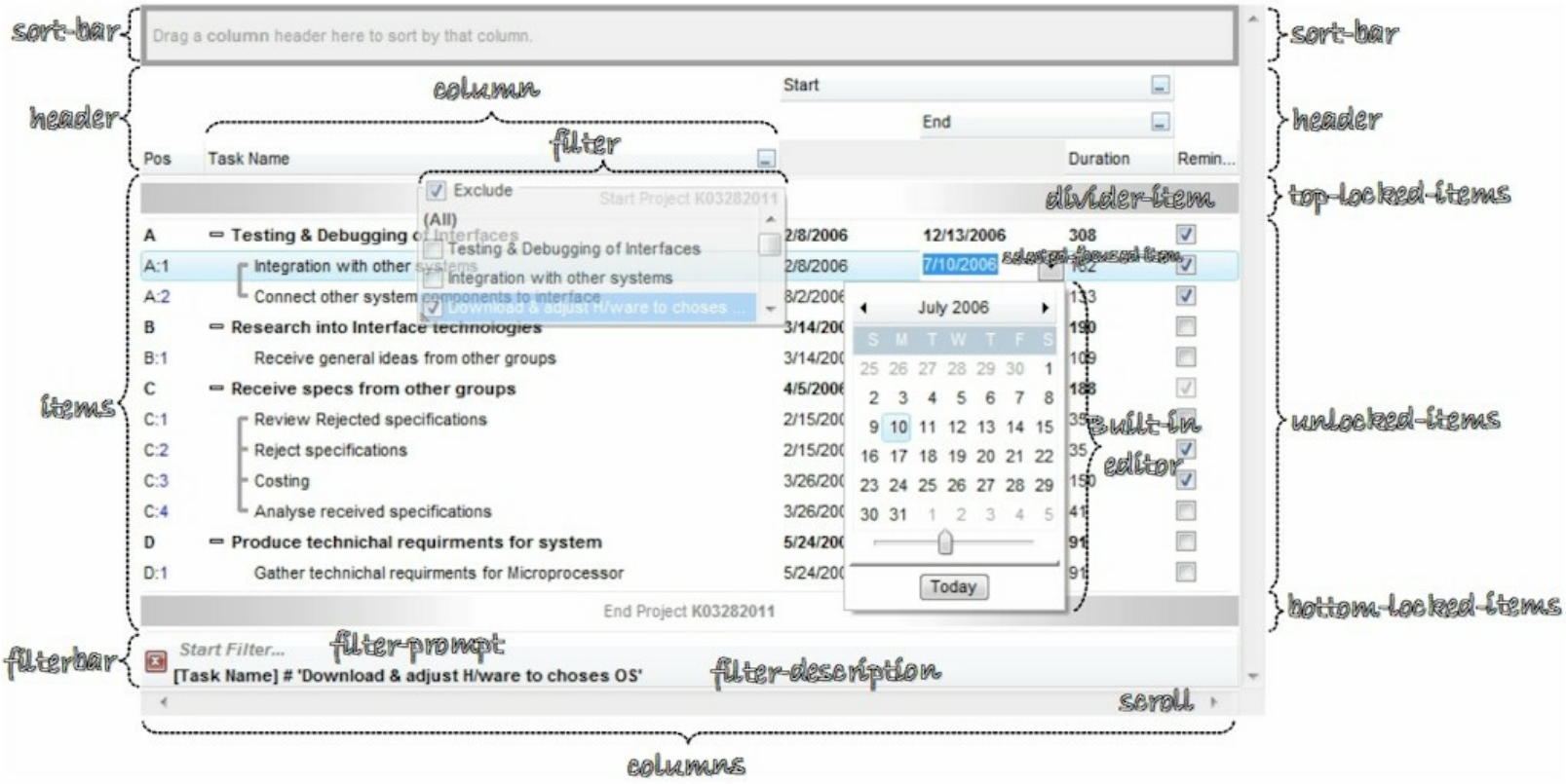
Don't forget that you can contact our development team if you have ideas or requests for new components, by sending us an e-mail at support@exontrol.com (please include the name of the product in the subject, ex: exgrid) . We're sure our team of developers will try to find a way to make you happy - and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

Regards,
Exontrol Development Team

https://www.exontrol.com

How to start?

The following screen shot, shows a general idea how parts and objects of the control are arranged:



click to enlarge

The following steps shows you progressively how to start programming the Exontrol's ExGrid component:

- **Load / Save Data**. The control provides several ways to serialize your data, as listed:
  - LoadXML / SaveXML methods, to load / save data using XML format.
  - DataSource property, to load / update / save data from a table, query, dataset and so on.
  - GetItems / PutItems methods, to load / save data from a/to safe array of data.

  For instance,

```
With Grid1
   .LoadXML "https://www.exontrol.net/testing.xml"
End With
```

loads control's data from specified URL.

- **Columns**. The control supports multiple columns, so always you can add / remove / move / hide any column

  - Add method, adds a new column.
  - ExpandColumns property specifies the columns to be shown/hidden when the column is expanded or collapsed.

  For instance,

```
With Grid1
   With .Columns.Add("Check")
      .Position = 0
      .Def(exCellHasCheckBox) = True
   End With
End With
```

  adds a new column that displays check-boxes, and that's the first visible column.

- **Editors**. Any cell / column of the control supports built-in editors, that let user edits data

  - EditType method, specifies the built-in to be assigned to a cell or column.
  - Editor property, gets access to the column's built-in editor
  - CellEditorVisible property specifies the built-in editor for a particular cell.

  For instance,

```
With Grid1
   With .Columns.Add("Date")
      .Editor.EditType = DateType
   End With
End With
```

  adds a new column that displays and edits column's data as date type.

- **Items**. Any item can hold a collection of child items. Any item is divided in cells, once cell for each column in the control.

  - AddItem method, adds a new item.
  - InsertItem method, inserts a child item
  - InsertControlItem method, inserts a child item that hosts another control

inside.

For instance,

```
With Grid1
   With .Items
      .AddItem "new item"
   End With
End With
```

adds a new item.

- **Cells**. An item contains a collection of cells, one cell for each column in the control. Any cell can be split or merge with one or more neighbor cells.

  - CellValue property, specifies the cell's value.

  For instance,

```
With Grid1
   With .Items
      h = .InsertItem(.FocusItem,"","item 1.1")
      .CellValue(h,1) = "item 1.2"
      .CellValue(h,2) = "item 1.3"
      .ExpandItem(.FocusItem) = True
   End With
End With
```

  adds a new child item of the focused item, and fills the cell's value for the second and third column.

Send comments on this topic.

constants AlignmentEnum

Specifies the source's alignment. Use the Alignment property to specify the column's alignment. Use the CellHAlignment property to specify the cell's alignment.

Name	Value	Description
LeftAlignment	0	The source is left aligned
CenterAlignment	1	The source is center aligned
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	The source has no borders.
Flat	1	Flat border
Sunken	2	Sunken border
Raised	3	Raised border
Etched	4	Etched border
Bump	5	Bump border

constants ArrowHandleEnum

The ArrorHandleEnum expression specifies the options for exLeftArrow, exRightArrow, exDownArrow or exUpArrow values when the Option property is used.

Name	Value	Description
exHandleEditor	0	The editor handles the arrow key. The key moves the cursor, if exists, inside the edit control. If the editor displays a caret, the F2 key selects or unselects the entire text.
exHandleControl	-1	The control handles the arrow key. The key moves the focus to a new cell. If the editor displays a caret, the F2 key selects or unselects the entire text. If the entire text is selected the key moves the focus to a new cell. If the text is not fully selected, the key moves the cursor to the next position, and if it is not available the next cell is focused.
exHandleEditSel	1	The editor handles the arrow key. The key moves the focus to a new cell, if the editor displays a caret and the key is pressed. If the text is not fully selected, the key moves the caret inside the editor. The F2 key selects or unselects the text inside the editor.

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. Use the ContinueColumnScroll property on True to allow scrolling the columns pixel by pixel. Click here or to watch a movie on how exAutoDragScroll works.
		exAutoDragPositionOnShortTouch. The object can

exAutoDragPositionOnShortTouch	256	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	655360	exAutoDragCopyImageOnRight. Drag and drop the selected objects to a target application, and paste them as image only.
exAutoDragCopySnapShotOnRight	720896	exAutoDragCopySnapShotOnRight. Drag and drop a snap shot of the current component.
exAutoDragScrollOnRight	1048576	exAutoDragScrollOnRight. The component is scrolled by clicking the object and dragging to a new position.
exAutoDragPositionOnLongTouch	16777216	exAutoDragPositionOnLongTouch. The object can be dragged from a position to another, but not outside of its group.
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	268435456	exAutoDragCopyTextOnLongTouch. Drag and drop the selected objects to a target application, and paste them as text only.
exAutoDragCopyImageOnLongTouch	536870912	exAutoDragCopyImageOnLongTouch. Drag and drop the selected objects to a target application, and paste them as image only.
exAutoDragCopySnapShotOnLongTouch	1073741824	exAutoDragCopySnapShotOnLongTouch. Drag and drop a snap shot of the current component.
exAutoDragScrollOnLongTouch	2147483648	exAutoDragScrollOnLongTouch. The component is scrolled by clicking the object and dragging to a new position.

constants AutoSearchEnum

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

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

constants BackgroundPartEnum

The BackgroundPartEnum type indicates parts in the control. Use the Background property to specify a background color or a visual appearance for specific parts in the control. A Color expression that indicates the background color for a specified part. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

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

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

Name	Value	Description
exHeaderFilterBarButton	0	Specifies the background color for the drop down filter bar button. 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. By default, the exCellButtonUp property is 0, which indicates that the current visual theme is responsible to show the button's face while it is up. Use the CellHasButton property to assign a button to a cell. The exCursorHoverCellButton property specifies the

		visual appearance for the cell's button when the cursor hovers it.
exCellButtonDown	3	Specifies the background color for the cell's button, when it is down. By default, the exCellButtonDown property is 0, which indicates that the current visual theme is responsible to show the button's face while it is down. Use the CellHasButton property to assign a button to a cell. The exCursorHoverCellButton property specifies the visual appearance for the cell's button when the cursor hovers it.
exDropDownButtonUp	4	Specifies the visual appearance for the drop down button, when it is up. Usually the editors with a drop down portion displays a drop down button.
exDropDownButtonDown	5	Specifies the visual appearance for the drop down button, when it is down. Usually the editors with a drop down portion displays a drop down button.
exButtonUp	6	Specifies the visual appearance for the button inside the editor, when it is up. Use the AddButton method to add new buttons to an editor.
exButtonDown	7	Specifies the visual appearance for the button inside the editor, when it is down. Use the AddButton method to add new buttons to an editor.
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.
		Specifies the visual appearance for the slider's bar.

exSliderRange	15	The SliderType specifies whether an editor displays a slider bar control.
exSliderThumb	16	Specifies the visual appearance for the thumb of the slider. The SliderType specifies whether an editor displays a slider bar control.
exSelectInPlace	17	Specifies the visual appearance for the selection when a drop down editor is focused and closed.
exShowFocusRect	19	Specifies the visual appearance to display the cell with the focus.
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. The SpinType or SliderType editor may display spin controls.
exSpinUpButtonUp	22	Specifies the visual appearance for the up spin button when it is not pressed.
exSpinUpButtonDown	23	Specifies the visual appearance for the up spin button when it is pressed.
exSpinDownButtonUp	24	Specifies the visual appearance for the down spin button when it is not pressed.
exSpinDownButtonDown	25	Specifies the visual appearance for the down spin button when it is pressed.
exBackColorFilter	26	Specifies the background color for 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.
exCursorHoverColumn	32	Specifies the visual appearance for the column when the cursor hovers it (header column). By default, the exCursorHoverColumn property is zero, which indicates that the current theme is responsible to show the column while cursor hovers it. If exCursorHoverColumn property is -1, no visual

		appearance is changed while the cursor hovers the column's 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. 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.
exDragDropAlign	40	
exHeaderFilterBarActive	41	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 floating bar. The ColumnsFloatBarVisible property indicates whether the hidden columns are shown to a floating panel. The exColumnsFloatAppearance property can be used to change the floating panel's frame.
exColumnsFloatScrollBackColor	88	Specifies the background color for the scroll bars in the Columns float bar. The ColumnsFloatBarVisible property indicates whether the hidden columns are shown to a floating panel.
		Specifies the background color for the scroll bars in

exColumnsFloatScrollPressBackColor	89	the Columns float bar, while the scroll part is pressed. The ColumnsFloatBarVisible property indicates whether the hidden columns are shown to a floating panel.
exColumnsFloatScrollUp	90	Specifies the visual appearance of the up scroll bar. The ColumnsFloatBarVisible property indicates whether the hidden columns are shown to a floating panel.
exColumnsFloatScrollDown	91	Specifies the visual appearance of the down scroll bar. The ColumnsFloatBarVisible property indicates whether the hidden columns are shown to a floating panel.
exColumnsFloatAppearance	92	Specifies the visual appearance for the frame/borders of the Column's float bar The ColumnsFloatBarVisible property indicates whether the hidden columns are shown to a floating panel.
exColumnsFloatCaptionBackColor	93	Specifies the visual appearance for caption, if the Background(exColumnsFloatAppearance) property is specified. The ColumnsFloatBarVisible property indicates whether the hidden columns are shown to a floating panel.
exColumnsFloatCaptionForeColor	94	Specifies the foreground color for the caption, if the Background(exColumnsFloatAppearance) property is specified. The ColumnsFloatBarVisible property indicates whether the hidden columns are shown to a floating panel.
exColumnsFloatCloseButton	95	Specifies the visual appearance for the closing button, if the Background(exColumnsFloatAppearance) property is specified. The ColumnsFloatBarVisible property indicates whether the hidden columns are shown to a floating panel.
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.
exCursorHoverCellButton	157	Specifies the visual appearance for the cell's button when the cursor hovers it. By default, the exCursorHoverCellButton property is zero, which indicates that the current theme is responsible to show the cell's button while cursor hovers it. If exCursorHoverCellButton property is -1, no visual appearance is changed while the cursor hovers the cell's button.
exSelBackColorHide	166	Specifies the selection's background color, when the control has no focus.
exSelForeColorHide	167	Specifies the selection's foreground color, when the control has no focus.
exTreeGlyphOpen	180	Specifies the visual appearance for the +/- buttons when it is collapsed.
exTreeGlyphClose	181	Specifies the visual appearance for the +/- buttons when it is expanded.
exColumnsPositionSign	182	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	Specifies the color to show the tree-lines (connecting lines from the parent to the children)
exVSUp	256	The up button in normal state.
exVSUpP	257	The up button when it is pressed.
exVSUpD	258	The up button when it is disabled.
exVSUpH	259	The up button when the cursor hovers it.
exVSThumb	260	The thumb part (exThumbPart) in normal state.
exVSThumbP	261	The thumb part (exThumbPart) when it is pressed.
exVSThumbD	262	The thumb part (exThumbPart) when it is disabled.
exVSThumbH	263	The thumb part (exThumbPart) when cursor hovers it.
exVSDown	264	The down button in normal state.

exVSDownP	265	The down button when it is pressed.
exVSDownD	266	The down button when it is disabled.
exVSDownH	267	The down button when the cursor hovers it.
exVSLower	268	The lower part (exLowerBackPart) in normal state.
exVSLowerP	269	The lower part (exLowerBackPart) when it is pressed.
exVSLowerD	270	The lower part (exLowerBackPart) when it is disabled.
exVSLowerH	271	The lower part (exLowerBackPart) when the cursor hovers it.
exVSUpper	272	The upper part (exUpperBackPart) in normal state.
exVSUpperP	273	The upper part (exUpperBackPart) when it is pressed.
exVSUpperD	274	The upper part (exUpperBackPart) when it is disabled.
exVSUpperH	275	The upper part (exUpperBackPart) when the cursor hovers it.
exVSBack	276	The background part (exLowerBackPart and exUpperBackPart) in normal state.
exVSBackP	277	The background part (exLowerBackPart and exUpperBackPart) when it is pressed.
exVSBackD	278	The background part (exLowerBackPart and exUpperBackPart) when it is disabled.
exVSBackH	279	The background part (exLowerBackPart and exUpperBackPart) when the cursor hovers it.
exHSLeft	384	The left button in normal state.
exHSLeftP	385	The left button when it is pressed.
exHSLeftD	386	The left button when it is disabled.
exHSLeftH	387	The left button when the cursor hovers it.
exHSThumb	388	The thumb part (exThumbPart) in normal state.
exHSThumbP	389	The thumb part (exThumbPart) when it is pressed.
exHSThumbD	390	The thumb part (exThumbPart) when it is disabled.

exHSThumbH	391	The thumb part (exThumbPart) when the cursor hovers it.
exHSRight	392	The right button in normal state.
exHSRightP	393	The right button when it is pressed.
exHSRightD	394	The right button when it is disabled.
exHSRightH	395	The right button when the cursor hovers it.
exHSLower	396	The lower part (exLowerBackPart) in normal state.
exHSLowerP	397	The lower part (exLowerBackPart) when it is pressed.
exHSLowerD	398	The lower part (exLowerBackPart) when it is disabled.
exHSLowerH	399	The lower part (exLowerBackPart) when the cursor hovers it.
exHSUpper	400	The upper part (exUpperBackPart) in normal state.
exHSUpperP	401	The upper part (exUpperBackPart) when it is pressed.
exHSUpperD	402	The upper part (exUpperBackPart) when it is disabled.
exHSUpperH	403	The upper part (exUpperBackPart) when the cursor hovers it.
exHSBack	404	The background part (exLowerBackPart and exUpperBackPart) in normal state.
exHSBackP	405	The background part (exLowerBackPart and exUpperBackPart) when it is pressed.
exHSBackD	406	The background part (exLowerBackPart and exUpperBackPart) when it is disabled.
exHSBackH	407	The background part (exLowerBackPart and exUpperBackPart) when the cursor hovers it.
exSBtn	324	All button parts (L1-L5, LButton, exThumbPart, RButton, R1-R6), in normal state.
exSBtnP	325	All button parts (L1-L5, LButton, exThumbPart, RButton, R1-R6), when it is pressed.
exSBtnD	326	All button parts (L1-L5, LButton, exThumbPart, RButton, R1-R6), when it is disabled.
exSBtnH	327	All button parts (L1-L5, LButton, exThumbPart,

		RButton, R1-R6), when the cursor hovers it .
exScrollHoverAll	500	Enables or disables the hover-all feature. By default (Background(exScrollHoverAll) = 0), the left/top, right/bottom and thumb parts of the control' scrollbars are displayed in hover state while the cursor hovers any part of the scroll bar (hover-all feature). The hover-all feature is available on Windows 11 or greater, if only left/top, right/bottom, thumb, lower and upper-background parts of the scrollbar are visible, no custom visual-appearance is applied to any visible part. The hover-all feature is always on If Background(exScrollHoverAll) = -1. The Background(exScrollHoverAll) = 1 disables the hover-all feature.
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	Specifies the visual appearance of the control's size grip when both scrollbars are shown.

constants BackModeEnum

Specifies the background mode when painting the selected items. Use the SelBackMode property to specify the control's selection back mode.

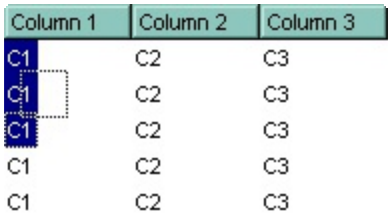
Name	Value	Description
exOpaque	0	The selection is opaque.
exTransparent	1	The selection is transparent.
exGrid	2	The selection is half transparent half opaque

constants CellSelectEnum

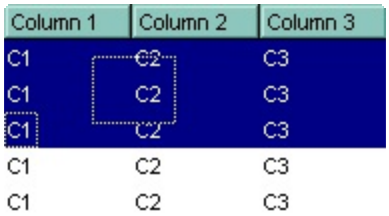
Specifies how the control selects cells or items within the control. Use the FullRowSelect property to enables full-row selection.

Name	Value	Description
exColumnSel	0	(**False**) Enables single-cell selection in the control.
exItemSel	-1	(**True**) Enables full-row selection in the control.
exRectSel	1	Enables rectangle selection in the control.

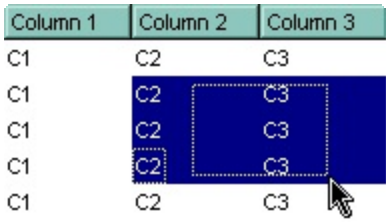
When the FullRowSelect property is **exColumnSel** the selection looks like:



When the FullRowSelect property is **exItemSel** the selection looks like:



When the FullRowSelect property is **exRectSel** the selection looks like:



constants CellSingleLineEnum

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

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

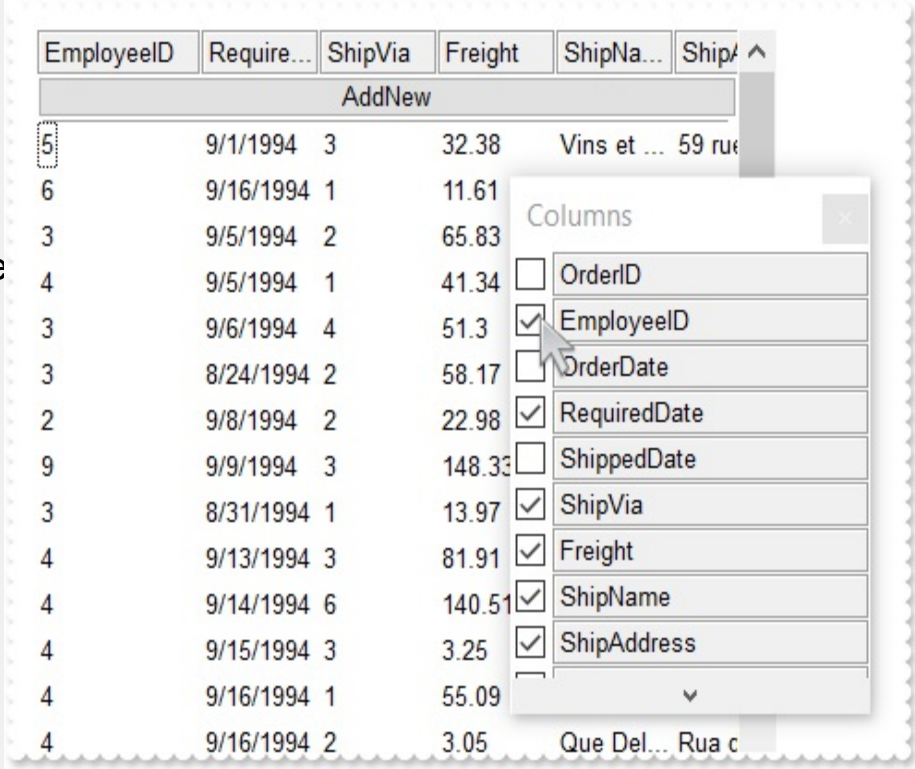
constants CheckStateEnum

Specifies the cell's state if CellHasCheckBox or CellHasRadioButton property is True.

Name	Value	Description
Unchecked	0	The cell is not checked.
Checked	1	The cell is checked.
PartialChecked	2	The cell is partially checked. To allow partially checks for a cell, the PartialCheck property should be True.

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.
exColumnsFloatBarVisibleIncludeGroupByColumns	1	Specifies that the control's Columns float-panel shows only columns that can be group- by (dragable-columns only). The AllowGroupBy property specifies whether the column can be group-by.
exColumnsFloatBarVisibleIncludeChecks	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)*
exCellValueFormat	17	Similar with the CellValueFormat property, *(ValueFormatEnum expression, exText)*
		Specifies the format layout for the cells. The CellFormatLevel property indicates the format layout for a specified cell. Use the FormatLevel

exCellFormatLevel	32	property to specify the layout of the column in the control's header bar. *(CRD string expression)*
exCellOwnerDraw	33	Assigns an owner draw object for the entire column. Use the CellOwnerDraw property to assign an owner draw object to a single cell. *(an object that implements the IOwnerDrawHandler interface)*
exCellDrawPartsOrder	34	Specifies the order of the drawing parts for the entire column. By default, this option is "check,icon,icons,picture,caption", which means that the cell displays its parts in the following order: check box/ radio buttons (CellHasCheckBox/CellRadioButton), single icon (CellImage), multiple icons (CellImages), custom size picture (CellPicture), and the cell's caption. Use the exCellDrawPartsOrder option to specify a new order for the drawing parts in the cells of the column. The RightToLeft property automatically flips the order of the columns. *(String expression, "check,icon,icons,picture,caption")*
exCellPaddingLeft	48	Gets or sets the left padding (space) of the cells within the column. *(Long expression)*
exCellPaddingRight	49	Gets or sets the right padding (space) of the cells within the column. *(Long expression)*
exCellPaddingTop	50	Gets or sets the top padding (space) of the cells within the column. *(Long expression)*

exCellPaddingBottom	51	Gets or sets the bottom padding (space) of the cells within the column. *(Long expression)*
exHeaderPaddingLeft	52	Gets or sets the left padding (space) of the column's header. *(Long expression)*
exHeaderPaddingRight	53	Gets or sets the right padding (space) of the column's header. *(Long expression)*
exHeaderPaddingTop	54	Gets or sets the top padding (space) of the column's header. *(Long expression)*
exHeaderPaddingBottom	55	Gets or sets the bottom padding (space) of the column's header. *(Long expression)*
exColumnResizeContiguously	64	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 several 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 in the drop down filter window is not shown.
exFilterBarBlanks	1	Defines the caption of (Blanks) in the filter bar window. If the Description(exFilterBarBlanks) property is empty, the (Blanks) predefined item in the drop down filter window is not shown.
exFilterBarNonBlanks	2	Defines the caption of (NonBlanks) in the filter bar window. If the Description(exFilterBarNonBlanks) property is empty, the (NonBlanks) predefined item in the drop down filter window is not shown.
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 the filter window when it displays no pattern field.
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

exFilterBarDateTo	13	down filter window. For instance, the "to 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. The ColumnsFloatBarVisible property specifies whether the hidden columns float bar is visible or hidden.

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. Use the ItemDivider property to define a divider item.

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

constants DropDownWidthType

The DropDownWidthType expression specifies the width of the drop down portion of an editor. The DropDownAutoWidth property specifies the width of the drop down portion of the editor. The DropDownMinWidth property specifies the minimum width for the drop down portion.

Name	Value	Description
exDropDownAutoWidth	-1	The drop down width is automatically computed to let all predefined items in the editor fi the drop down portion.
exDropDownEditorWidth	0	The width of the drop down portion of the editor is specified by the width of the cell that holds the editor.
exDropDownAutoEditorWidth	1	The width of the drop down portion of the editor is specified by the width of the cell that holds the editor. The width of the drop down can't be less than the width required to let all predefined items being visible. The width of the drop down portion is always greater than the DropDownMinWidth value.

constants EditorOptionEnum

Specifies different options for a built-in editor. The Option property specifies the editor's options. Use the DefaultEditorOption property to specify default option for the editors of a specified type. The following options are supported:

Name	Value	Description
exMemoHScrollBar	1	Adds the horizontal scroll bar to a MemoType or MemoDropDownType editor. *(boolean expression, by default it is false)*
exMemoVScrollBar	2	Adds the vertical scroll bar to a MemoType or MemoDropDownType editor. *(boolean expression, by default it is false)*
exMemoAutoSize	3	Specifies whether the MemoType editor is resized when user alters the text. *(boolean expression, by default it is true)*
exColorListShowName	4	Specifies whether a ColorListType editor displays the name of the color. *(boolean expression, by default it is false)*
exColorShowPalette	5	Specifies whether the ColorList editor displays the palette colors list. *(boolean expression, by default it is true)*
exColorShowSystem	6	Specifies whether the ColorType editor shows the system colors list. *(boolean expression, by default it is true)*
exMemoDropDownWidth	7	Specifies the width for a MemoDropDownType editor. *(long expression, by default it is 128)*

exMemoDropDownHeight	8	Specifies the height for a MemoDropDownType editor. *(long expression, by default it is 116)*
exMemoDropDownAcceptReturn	9	Specifies whether the Return key is used to add new lines into a MemoDropDownType editor. *(boolean expression, by default it is true)*
exEditRight	10	Right-aligns text in a single-line or multiline edit control. *(boolean expression, by default it is false)*
exProgressBarBackColor	11	Specifies the background color for a progress bar editor. Use the exProgressBarMarkTicker option to specify the background color or visual appearance of the progress bar. *(color expression, by default it is 0x80000000 | COLOR_HIGHLIGHT)*
exProgressBarAlignment	12	Specifies the alignment of the caption inside of a progress bar editor. *(AlignmentEnum expression, by default it is LeftAlignment)*
exProgressBarMarkTicker	13	Retrieves or sets a value that indicates whether the ticker of a progress bar editor is visible or hidden. If value is 0 (false), no progress's background is shown. If -1(true), the progress's background is shown using the current visual theme, else the solid color or the EBN object is applied on the progress's background. *(color expression, by default it is -1)*
exDateAllowNullDate	14	Allows you to specify an empty date to a DateType editor. *(boolean expression, by default it is true)*

exCheckValue0	15	Specifies the check box state being displayed for unchecked state. *(long expression, valid values are 0, 1 or 2, by default it is 0)*
exCheckValue1	16	Specifies the check box state being displayed for checked state. *(long expression, valid values are 0, 1 or 2, by default it is 1)*
exCheckValue2	17	Specifies the check box state being displayed for partial checked state. (long expression, valid values are 0, 1 or 2). For instance, if your cells load boolean values (True is -1, False is 0), the control displays the partial-check icon for True values. You can call Grid1.DefaultEditorOption(exCheckValue2) = 1 before loading the CheckValueType editor, and so the partial-check cells show as check icons. *(long expression, valid values are 0, 1 or 2, by default it is 2)*
exEditPassword	18	Specifies a value that indicates whether an edit control displays all characters as an asterisk (*) as they are typed (passwords). *(boolean expression, by default it is false)*
exEditPasswordChar	19	Specifies a value that indicates the password character. *(character expression, by default it is '*')*
exLeftArrow	20	(VK_LEFT) Specifies whether the left arrow key is handled by the control or by the editor. By default, the Option(exLeftArrow) property is exHandleControl. Use the exLeftArrow option to disable focusing a new cell if the user presses the left arrow key while editing. The option is valid for

		all editors. *(ArrowHandleEnum expression, by default it is exHandleControl)*
exRightArrow	21	(VK_RIGHT) Specifies whether the right arrow key is handled by the control or by the editor. By default, the Option(exRightArrow) property is exHandleControl. Use the exRightArrow option to disable focusing a new cell if the user presses the right arrow key while editing. The option is valid for all editors. *(ArrowHandleEnum expression, by default it is exHandleControl)*
exUpArrow	22	(VK_UP) Specifies whether the up arrow key is handled by the control or by the editor. By default, the Option(exUpArrow) property is exHandleControl. Use the exUpArrow option to disable focusing a new cell if the user presses the up arrow key while editing. The option is valid for all editors. *(ArrowHandleEnum expression, by default it is exHandleControl)*
exDownArrow	23	(VK_DOWN) Specifies whether the down arrow key is handled by the control or by the editor. By default, the Option(exDownArrow) property is exHandleControl. Use the exDownArrow option to disable focusing a new cell if the user presses the down arrow key while editing. The option is valid for all editors. *(ArrowHandleEnum expression, by default it is exHandleControl)*
		(VK_HOME) Specifies whether the home key is handled by the control or by the current editor. By default, the Option(exHomeKey) property is True. Use the exHomeKey option to disable focusing a new cell if the user presses the home key while

exHomeKey	24	editing. The option is valid for all editors. *(boolean expression, by default it is true)*
exEndKey	25	(VK_END) Specifies whether the end key is handled by the control or by the current editor. By default, the Option(exEndKey) property is True. Use the exEndKey option to disable focusing a new cell if the user presses the end key while editing. The option is valid for all editors. *(boolean expression, by default it is true)*
exPageUpKey	26	(VK_PRIOR) Specifies whether the page up key is handled by the control or by the current editor. By default, the Option(exPageUpKey) property is True. Use the exPageUpKey option to disable focusing a new cell if the user presses the page up key while editing. The option is valid for all editors. *(boolean expression, by default it is true)*
exPageDownKey	27	(VK_NEXT) Specifies whether the page down key is handled by the control or by the current editor. By default, the Option(exPageDownKey) property is True. Use the exPageDownKey option to disable focusing a new cell if the user presses the page down key while editing. The option is valid for all editors. *(boolean expression, by default it is true)*
exDropDownImage	28	Displays the predefined icon in the control's cell, if the user selects an item from a drop down editor. By default, the exDropDownImage property is True. The option is valid for DropDownListType, PickEdit and ColorListType editors. *(boolean expression, by default it is true)*
		Specifies the caption for the 'Today' button in a

exDateTodayCaption	29	DateType editor. *(string expression, by default it is "Today")*
exDateMonths	30	Specifies the name for months to be displayed in a DateType editor. The list of months should be delimitated by spaces. *(string expression, by default it is "January February March April May June July August September October November December")*
exDateWeekDays	31	Specifies the shortcut for the weekdays to be displayed in a DateType editor. The list of shortcut for the weekdays should be separated by spaces. The first shortcut in the list indicates the shortcut for the Sunday, the second shortcut indicates the shortcut for Monday, and so on. *(string expression, by default it is ""S M T W T F S")*
exDateFirstWeekDay	32	Specifies the first day of the week in a DateType editor. The valid values for the Editor.Option(exDateFirstWeekDay) property are like follows: 0 - Sunday, 1 - Monday, 2 - Tuesday, 3 - Wednesday, 4 - Thursday, 5 - Friday and 6 - Saturday. *(long expression, valid values are 0 to 6, by default it is 0)*
exDateShowTodayButton	33	Specifies whether the 'Today' button is visible or hidden in a DateType editor. *(boolean expression, by default it is true)*
exDateMarkToday	34	Gets or sets a value that indicates whether the today date is marked in a DateType editor. *(boolean expression, by default it is false)*

exDateShowScroll	35	Specifies whether the years scroll bar is visible or hidden in a DateType editor. *(boolean expression, by default it is true)*
exEditLimitText	36	Limits the length of the text that the user may enter into an edit control. By default, the Editor.Option(exEditLimitText) is zero, and so no limit is applied to the edit control. *(long expression, by default it is 0)*
exAutoDropDownList	37	The exAutoDropDownList has no effect Editor.Option(exAutoDropDownList) property is 0 (default). Automatically shows the drop down list when user starts typing characters into a DropDownList editor, if the Editor.Option(exAutoDropDownList) property is -1. If the Editor.Option(exAutoDropDownList) property is +1, the control selects a new item that matches typed characters without opening the drop down portion of the editor. *(long expression, valid values are -1, 0 and +1, by default it is 0)*
exExpandOnSearch	38	Expands items while user types characters into a drop down editor. The exExpandOnSearch type has effect for drop down type editors. *(boolean expression, by default it is false)*
exAutoSearch	39	Specifies the kind of searching while user types characters within the drop down editor. The exExpandOnSearch type has effect for drop down type editors. *(AutoSearchEnum expression, valid values are 0 and 1, by default it is exStartWith)*
		Specifies the proposed change when user clicks a spin control. The exSpinStep should be a positive number, else clicking the spin has no effect. Integer

exSpinStep	40	or floating points allowed as well. For instance, if the exSpinStep is 0.01, the proposed change when user clicks the spin is 0.01. If the exSpinStep property is 0, the spin control is hidden (useful if you have a slider control). *(positive numeric expression, by default it is 1)*
exSliderWidth	41	Specifies the width in pixels of the slider control. The exSliderWidth value could be 0, when the slider control is hidden, a positive value that indicates the width in pixels of the slider in the control, a negative number when its absolute value indicates the percent of the cell's size being used by the slider. For instance, Option(exSliderWidth) = 0, hides the slider, Option(exSliderWidth) = 100, shows a slider of 100 pixels width, Option(exSliderWidth) = -50, uses half of the cell's client area to display a slider control. By default the Option(exSliderWidth) property is 64 pixels. Use the exSpinStep to hide the spin control. *(long expression, by default it is 64)*
exSliderStep	42	Specifies a value that represents the proposed change in the slider control's position. The exSliderTickFrequency property specifies the frequency to display ticks on a slider control. The exSliderMin and exSliderMax determines the range of values for the slider control. *(numeric expression , by default it is 1)*
exSliderMin	43	Specifies the slider's minimum value. *(numeric expression, by default it is 0)*
exSliderMax	44	Specifies the slider's maximum value. *(numeric expression, by default it is 100)*
		Keeps the selection background color while the

exKeepSelBackColor	45	editor is visible. The exKeepSelBackColor option is valid for all editors. Use the exKeepSelBackColor to let the editor to display the control's selection background color when it is visible. *(boolean expression, by default it is false)*
exEditDecimalSymbol	46	Specifies the symbol that indicates the decimal values while editing a floating point number. Use the exEditDecimaSymbol option to assign a different symbol for floating point numbers, when Numeric property is exFloat. *(long expression, that indicates the ASCII code for the character being used as decimal symbol, by default, it is the "Decimal symbol" settings as in the Regional Options, in your control panel)*
exDateWeeksHeader	47	Sets or gets a value that indicates whether the weeks header is visible or hidden in a DateType editor. *(boolean expression, by default it is false)*
exEditSelStart	48	Sets the starting point of text selected, when an EditType editor is opened. If the exEditSelStart property is 0, the text gets selected from the first character. If the exEditSelStart property is -1, the cursor is placed at the end of the text. *(long expression, by default it is 0)*
exEditSelLength	49	Sets the number of characters selected, when an EditType editor is opened. If the exEditSelLength is 0, no text is selected, instead the exEditSelStart changes the position of the cursor. If the exEditSelLength property is -1, the text from the exEditSelStart position to the end gets selected. *(long expression, by default it is -1)*
		Specifies the background color for a locked edit

exEditLockedBackColor	50	control. By default, the exEditLockedBackColor property is a system color that indicates the face color for three-dimensional display elements and for dialog box backgrounds. *(color expression, by default it is 0x80000000 | COLOR_3DFACE)*
exEditLockedForeColor	51	Specifies the foreground color for a locked edit control. *(color expression, by default it is 0)*
exShowPictureType	52	Specifies whether a PictureType editor displays the type of the picture. *(boolean expression, by default it is true)*
exSliderTickFrequency	53	Gets or sets the interval between tick marks slider types. By default, the exSliderTickFrequency property is 0 which makes the slider to display no ticks. The exSliderTickFrequency property specifies the frequency to display ticks on a slider control. The exSliderStep proposed change in the slider control's position. The exSliderMin and exSliderMax determines the range of values for the slider control. The exSliderWidth option specifies the width of the slider within the cell. *(numeric expression, by default it is 0)*
exPickAllowEmpty	54	Specifies whether the editor of PickEditType supports empty value. *(boolean expression, by default it is false)*
exDropDownBackColor	55	Specifies the drop down's background color. If 0 the exDropDownBackColor has no effect. *(color expression, by default it is 0)*
		Specifies the drop down's foreground color. If 0 the

exDropDownForeColor	56	exDropDownBackColor has no effect. *(color expression, by default it is 0)*
exDropDownColumnCaption	57	Specifies the HTML caption for each column within the drop down list, separated by Ś character (vertical broken bar, ALT + 221). For instance, "<sha ;;0>Name</sha>Ś<sha ;;0>ID</sha>" defines two columns for the drop down editor. The header of the drop down list is visible, if the exDropDownColumnCaption is not empty. *(string expression, by default it is "")*
exDropDownColumnWidth	58	Specifies the width for each column within the drop down list, separated by Ś character (vertical broken bar, ALT + 221). For instance, "Ś32" defines the width of the second column to 32 pixels, within a drop down multiple columns editor. *(string expression, by default it is "")*
exDropDownColumnPosition	59	Specifies the position for each column within the drop down list, separated by Ś character (vertical broken bar, ALT + 221). For instance, "Ś0" defines sets the second column to be first visible-column, within a drop down multiple columns editor. *(string expression, by default it is "")*
exDropDownColumnAutoResize	60	Specifies whether the drop down list resizes automatically its visible columns to fit the drop down width. Specifies whether the drop down multiple columns editor displays horizontal-scroll bar. *(boolean expression, by default it is true)*
exSliderTickStyle	63	exSliderTickStyle. Gets or sets the style to display the slider' ticks.
exCalcExecuteKeys	100	Specifies whether the calculator editor executes the keys while focused and the drop down portion is hidden.

		(boolean expression, by default it is true)
exCalcCannotDivideByZero	101	Specifies the message whether a division by zero occurs in a calendar editor. *(string expression, by default it is "Cannot divide by zero.")*
exCalcButtonWidth	102	Specifies the width in pixels of the buttons in the calculator editor. *(long expression, by default it is 24)*
exCalcButtonHeight	103	Specifies the height in pixels of the buttons in the calculator editor. *(long expression, by default it is 24)*
exCalcButtons	104	Specifies buttons in a calendar editor. The property specifies the buttons and the layout of the buttons in the control. A string expression that indicates the list of buttons being displayed. The rows are separated by chr(13)+chr(10) (vbCrLf) sequence, and the buttons inside the row are separated by ';' character. *(string expression)*
exCalcPictureUp	105	Specifies the picture when the button is up in a drop down calendar editor. A Picture object that indicates the node's picture. *(A Picture object that 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, by default it is "")*
		Specifies the picture when the button is down in a drop down calendar editor. A Picture object that indicates the node's picture.

exCalcPictureDown	106	*(A Picture object that 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, by default it is "")*
exEditAllowOverType	200	Specifies whether the editor supports overtype mode. The option is valid for EditType and MemoType editors. *(boolean expression, by default it is false)*
exEditOverType	201	Retrieves or sets a value that indicates whether the editor is in insert or overtype mode. The option is valid for EditType and MemoType editors. *(boolean expression, by default it is false)*
exEditAllowContextMenu	202	Specifies whether the editor displays the edit's default context menu when the user right clicks the field. *(boolean expression, by default it is true)*

constants EditorVisibleEnum

The EditorVisibleEnum type specifies the way the control shows the field's editor. The EnsureVisibleEnum type support the following values:

Name	Value	Description
exEditorHidden	0	The editor is hidden.
exEditorVisible	1	The editor is always visible.
exEditorVisibleOnFocus	-1	The editor is visible when the cell receives the focus.

constants EditTypeEnum

Use the EditType property to specify the editor for a cell or a column. Any editor can have a check box (use CellHasCheckBox property) , radio button (use CellHasRadioButton property) associated, or multiple buttons to the left or right side (use AddButton method). The Mask property is applied to most of all editors that has associated a standard edit control. Use the Option property to assign different options for a given editor. Use the DefaultEditorOption property to specify default option for the editors of a specified type. The CellValue property indicates the value for the editor. A cell or a column supports the following type of editors:

Name	Value	Description
ReadOnly	0	The column or the cell has no editor associated.
EditType	1	A standard text edit field.  The editor supports the following options: • exEditRight, Right-aligns text in a single-line or multiline edit control. • exEditPassword, Specifies a value that indicates whether an edit control displays all characters as an asterisk (*) as they are typed (passwords). • exEditPasswordChar, Specifies a value that indicates the password character. • exEditLimitText, Limits the length of the text that the user may enter into an edit control. • exEditDecimalSymbol, Specifies the symbol that indicates the decimal values while editing a floating point number. The Numeric property should be on exFloat. • exEditSelStart, Sets the starting point of text selected, when an EditType editor is opened. • exEditSelLength, Sets the number of characters selected, when an EditType editor is opened. • exEditLockedBackColor property. Specifies the background color for a locked edit control. • exEditLockedForeColor property. Specifies the foreground color for a locked edit control.
		It provides an intuitive interface for your

users to select values from pre-defined lists presented in a drop-down window, but it accepts new values at runtime too. The DropDownType editor has associated a standard text edit field too. Use AddItem or InsertItem method to add predefined values to the drop down list. The DropDownRows property specifies the maximum number of visible rows into the drop-down list. The editor displays the CellValue value, not the identifier of the selected item. The EditType options are supported too.

The following sample adds a column with a DropDownType editor:

DropDownType 2

```
With .Columns.Add("Editor").Editor
   .EditType = DropDownType
   .AddItem 0, "Single Bed", 1
   .AddItem 1, "Double Bed", 2
   .AddItem 2, "Apartment", 3
   .AddItem 3, "Suite", 4
   .AddItem 4, "Royal Suite", 5
End With
.Items.CellValue(.Items(0), "Editor") = "Apartment"
```

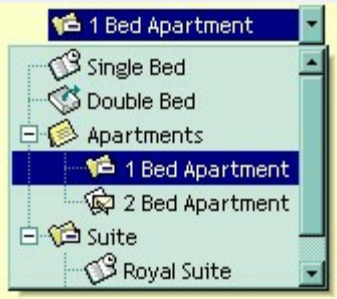
The editor supports the following options:

- exDropDownBackColor, specifies the drop down's background color
- exDropDownForeColor, specifies the drop down's foreground color
- exDropDownColumnCaption, specifies the HTML caption for each column within the drop down list, separated by Ś character (vertical broken bar, ALT + 221)
- exDropDownColumnWidth, specifies the width for each column within the drop down list, separated by Ś character (vertical broken bar, ALT + 221).
- exDropDownColumnPosition, specifies the position for each column within the drop down list, separated by Ś character (vertical broken

bar, ALT + 221).

- exDropDownColumnAutoResize, specifies whether the drop down list resizes automatically its visible columns to fit the drop down width

DropDownListType 3



It provides an intuitive interface for your users to select values from predefined lists presented in a drop-down window. The DropDownListType editor has no standard edit field associated. Use the AddItem or InsertItem method to add predefined values to the drop down list. The DropDownRows property specifies the maximum number of visible rows into the drop-down list. The editor displays the caption of the item that matches the CellValue value. The item's icon is also displayed if it exists.

The following sample adds a column with a DropDownListType editor:

```
With .Columns.Add("Editor").Editor
    .DropDownAutoWidth = False
    .EditType = DropDownListType
    .AddItem 0, "Single Bed", 1
    .AddItem 1, "Double Bed", 2
    .AddItem 2, "Apartments", 3
    .InsertItem 3, "1 Bed Apartment", 4, 2
    .InsertItem 4, "2 Bed Apartment", 5, 2
    .AddItem 5, "Suite", 4
    .InsertItem 6, "Royal Suite", 1, 5
    .InsertItem 7, "Deluxe Suite", 2, 5
    .ExpandAll
End With
.Items.CellValue(.Items(0), "Editor") = 3
```

The editor supports the following options:

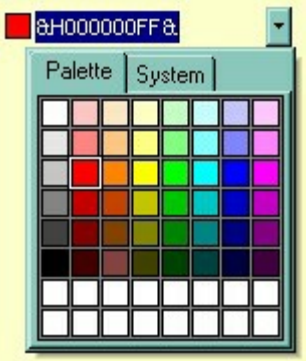
		• exDropDownImage, displays the predefined icon in the control's cell, if the user selects an item from a drop down editor. • exDropDownBackColor, specifies the drop down's background color • exDropDownForeColor, specifies the drop down's foreground color • exDropDownColumnCaption, specifies the HTML caption for each column within the drop down list, separated by Ś character (vertical broken bar, ALT + 221) • exDropDownColumnWidth, specifies the width for each column within the drop down list, separated by Ś character (vertical broken bar, ALT + 221). • exDropDownColumnPosition, specifies the position for each column within the drop down list, separated by Ś character (vertical broken bar, ALT + 221). • exDropDownColumnAutoResize, specifies whether the drop down list resizes automatically its visible columns to fit the drop down width
SpinType	4	The SpinType allows your users to view and change numeric values using a familiar up/down button (spin control) combination. The AddItem or InsertItem method has no effect, if the EditType is SpinType. Use the exSpinStep option to specify the proposed change when user clicks the spin. Use the Numeric property to specify whether the edit control allows only numeric values only. Use the exSpinUpButtonUp, exSpinUpButtonDown, exSpinDownButtonUp and exSpinDownButtonDown to change the visual appearance for the spin control.
		The MemoType is designed to provide an unique and intuitive interface, which you can implement within your application to assist users in working with textual

MemoType	5	information. If all information does not fit within the edit box, the window of the editor is enlarged. The AddItem or InsertItem method has no effect, if the EditType is MemoType. You can use options like exMemoHScrollBar, exMemoVScrollBar and so on.

CheckListType	6	It provides an intuitive interface for your users to check values from predefined lists presented in a drop-down window. Each item has a check box associated. The editor displays the list of item captions, separated by comma, that is OR combination of CellValue value. Use the The AddItem or InsertItem method to add new predefined values to the drop down list. The DropDownRows property specifies the maximum number of visible rows into the drop-down list. Use the CheckImage property to change the check box appearance.
		 
		
		The following sample adds a column with a CheckListType editor:
		`With .Columns.Add("Editor").Editor`
		`.EditType = CheckListType`
		`.AddItem 1, "Single Bed", 1`
		`.AddItem 2, "Double Bed", 2`
		`.AddItem 4, "Apartment", 3`
		`.AddItem 8, "Suite", 4`
		`.AddItem 16, "Royal Suite", 5`
		`End With`
		`.Items.CellValue(.Items(0), "Editor") = 5`
		The editor supports the following options:
		• exDropDownBackColor, specifies the drop down's background color
		• exDropDownForeColor, specifies the drop down's foreground color

DateType	7	 The DateType is a date/calendar control (not the Microsoft Calendar Control). The dropdown calendar provides an efficient and appealing way to edit dates at runtime. The DateType editor has a standard edit control associated. The user can easy select a date by selecting a date from the drop down calendar, or by typing directly the date. The editor displays the CellValue value as date. To change how the way how the control displays the date you can use FormatColumn event. The AddItem or InsertItem method has no effect, if the EditType is DateType. The following sample adds a column with a DateType editor: ``` With .Columns.Add("Editor").Editor .EditType = DateType End With .Items.CellValue(.Items(0), "Editor") = Date ```  You can use the MaskType to enter any data that includes literals and requires a mask to filter characters during data input. You can use this control to control the entry of many types of formatted information such as telephone numbers, social security numbers, IP addresses, license keys etc. The Mask property specifies the editor's mask. The MaskChar property specifies the masking character. The AddItem or InsertItem method has no effect, if the EditType is MaskType. The Mask property can use one or more literals: #,x,X,A,? <,>,*,\,{nMin,nMax},[...]. The following sample shows how to mask a column for input phone numbers: ``` With .Columns.Add("Editor").Editor ```
MaskType	8	

```
    .EditType = MaskType
    .Mask = "(###) ### - ####"
End With
.Items.CellValue(.Items(0), "Editor") = "(214) 345 - 789"
```

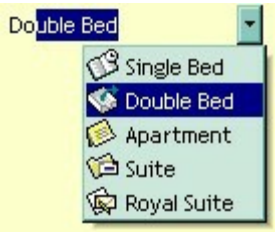
ColorType	9	 You can include a color selection control in your applications via the ColorType editor. Check the ColorListType also. The editor has a standard edit control and a color drop-down window. The color drop-down window contains two tabs that can be used to select colors, the "Pallette" tab shows a grid of colors, while the "System" tab shows the current windows color constants. The AddItem or InsertItem methodhas no effect, if the EditType is ColorType. You can use options like exColorShowPalette or exColorShowSystem.
		
		The following sample adds a column with a ColorType editor:
		
		`With .Columns.Add("Editor").Editor`
		`  .EditType = ColorType`
		`End With`
		`.Items.CellValue(.Items(0), "Editor") = vbRed`

FontType	10	 Provides an intuitive way for selecting fonts. The FontType editor contains a standard edit control and a font drop-down window. The font drop-down window contains a list with all system fonts. The AddItem or InsertItem method has no effect, if the EditType is FontType. The DropDownRows property specifies the maximum number of visible rows into the drop=down list.

The following sample adds a column with a FontType editor:

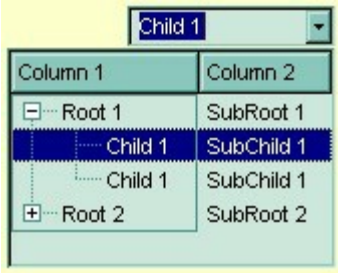
```
With .Columns.Add("Editor").Editor
    .EditType = FontType
End With
.Items.CellValue(.Items(0), "Editor") = "Times New Roman"
```

PictureType	11	 The PictureType provides an elegant way for displaying the fields of OLE Object type and cells that have a reference to an IPicture interface. An OLE Object field can contain a picture, a Microsoft Clip Gallery, a package, a chart, PowerPoint slide, a word document, a WordPad document, a wave file, an so on. In MS Access you can specify the field type to OLE Object. The DropDownMinWidth property specifies the minimum width for the drop=down window. The drop-down window is scaled based on the picture size. The AddItem or InsertItem method has no effect, if the EditType is PictureType. If your control is bounded to a ADO recordset, it automatically detects the OLE Object fields, so setting the editor's type to PictureType is not necessary. If your control is not bounded to an ADO recordset you can use the following sample to view OLE objects in the column "OLEObject" (the sample uses the NWIND database installed in your VB folder.
		
		Change the path if necessary, in the following sample:
		
		`' Creates an ADO Recordset`
		`Dim rs As Object`
		`Set rs = CreateObject("ADODB.Recordset")`
		`rs.Open "Employees",`

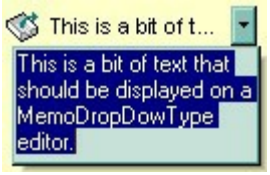
		"Provider=Microsoft.Jet.OLEDB.4.0;Data Source= D:\Program Files\Microsoft Visual Studio\VB98\NWIND.MDB", 3 ' Adds a column of PictureType edit Dim c As Column Set c = .Columns.Add("OLEObject") With c.Editor .EditType = PictureType End With **.Items.CellValue(.Items(0), "OLEObject") = rs("Photo").Value**
ButtonType	12	 The ButtonType editor consists into a standard edit field and a "..." button. The ButtonClick event is fired if the user has clicked the button. The AddItem or InsertItem method has no effect, if the EditType is ButtonType. Of course, you can apply for multiple buttons using the AddButton method, for any types.
ProgressBarType	13	 Uses the CellValue property to specify the percent being displayed in the ProgressBarTpe editor. The CellValue property should be between 0 and 100.
		 It provides an intuitive interface for your users to select values from pre-defined lists presented in a drop-down window. The PickEditType editor has a standard edit field associated, that useful for searching items while typing. The DropDownRows property specifies the maximum number of visible rows into the drop=down list. Use AddItem or InsertItem method to add new predefined values to the drop

PickEditType	14	down list. The editor displays the caption of the item that matches the CellValue value. The item's icon is also displayed if it exists.
		
		The following sample shows how to add values to a drop down list:
		
		`With .Columns.Add("Editor").Editor`
		`  .EditType = PickEditType`
		`  .AddItem 0, "Single Bed", 1`
		`  .AddItem 1, "Double Bed", 2`
		`  .AddItem 2, "Apartment", 3`
		`  .AddItem 3, "Suite", 4`
		`  .AddItem 4, "Royal Suite", 5`
		`End With`
		`.Items.CellValue(.Items(0), "Editor") = "Apartment"`
		
		The editor supports the following options:
		
		• exDropDownBackColor, specifies the drop down's background color
		• exDropDownForeColor, specifies the drop down's foreground color
		• exDropDownColumnCaption, specifies the HTML caption for each column within the drop down list, separated by Ś character (vertical broken bar, ALT + 221)
		• exDropDownColumnWidth, specifies the width for each column within the drop down list, separated by Ś character (vertical broken bar, ALT + 221).
		• exDropDownColumnPosition, specifies the position for each column within the drop down list, separated by Ś character (vertical broken bar, ALT + 221).
		• exDropDownColumnAutoResize, specifies whether the drop down list resizes automatically its visible columns to fit the drop down width

		The LinkEditType control allows your application to edit and display hyperlink

LinkEditType	15	addresses.
UserEditorType	16	 The control is able to use ActiveX controls as a built-in editor. The control uses the UserEditor property to define the user control. If it succeeded the UserEditorObject property retrieves the newly created object. Events like: UserEditOpen, UserEditClose and UserEditorOleEvent are fired when the control uses custom editors. The setup installs the VB\UserEdit, VC\User.Edit samples that uses Exontrol's ExComboBox component as a new editor into the ExGrid component (a multiple columns combobox control).
ColorListType	17	 You can include a color selection control in your application via the ColorListType editor, also. The editor hosts a predefined list of colors. By default. the following colors are added: Black, White, Dark Red, Dark Green, Dark Yellow, Dark Blue, Dark Magenta, Dark Cyan, Light Grey, Dark Grey, Red, Green, Yellow, Blue, Magenta, Cyan. The AddItem method adds a new color to your color list editor. You can use the exColorListShowName option to display the color's name. The following sample adds few custom colors to the ColorListType editor: ``` With .Columns.Add("Editor").Editor .EditType = ColorListType .AddItem 128, "Dark Red" ```

```
    .AddItem RGB(0, 128, 0), "Dark Green"
    .AddItem RGB(0, 0, 128), "Dark Blue"
End With
.Items.CellValue(.Items(0), "Editor") = 128
```

MemoDropDownType	18	It provides a multiple lines edit control that's displayed into a drop down window.  • The Editor.Option(exMemoDropDownWidth) specifies the width (in pixels) of the MemoDropDownType editor when it is dropped. • The Editor.Option(exMemoDropDownHeight) specifies the height (in pixels) of the MemoDropDownType editor when it is dropped. • The Editor.Option(exMemoDropDownAcceptReturn) specifies whether the user closes the MemoDropDownType editor by pressing the ENTER key. If the Editor.Option(exMemoDropDownAcceptReturn) is True, the user inserts new lines by pressing the ENTER key. The user can close the editor by pressing the CTRL + ENTER key. If the Editor.Option(exMemoDropDownAcceptReturn) is False, the user inserts new lines by pressing the CTRL + ENTER key. The user can close the editor by pressing the ENTER key. • The Editor.Option(exMemoHScrollBar) adds the horizontal scroll bar to a MemoType or MemoDropDownType editor. • The Editor.Option(exMemoVScrollBar) adds the vertical scroll bar to a MemoType or MemoDropDownType editor • Use the Items.CellSingleLine property to specify whether the cell displays multiple lines The AddItem or InsertItem method has no effect, if the EditType is MemoDropDownType.

CheckValueType	19	Displays check boxes in the column or cell. The CellValue property indicates the state of the cell's check box. See also: CellHasCheckBox property. The CheckValueType editor supports the following options: • exCheckValue0. Specifies the check box state being displayed for unchecked state • exCheckValue1. Specifies the check box state being displayed for checked state • exCheckValue2. Specifies the check box state being displayed for partial-check state For instance, if your cells load boolean values (True is -1, False is 0), the control displays the partial-check icon for True values. You can call the following code before loading the CheckValueType editor: `Grid1.DefaultEditorOption(exCheckValue2) = 1` in order to replace the partial-check appearance, to check state appearance.
SliderType	20	Adds a slider control to a cell. Use the exSliderWidth, exSliderStep, exSliderMin, exSliderMax options to control the slider properties. Use the exSpinStep option to hide the spin control. Use the exSpinUpButtonUp, exSpinUpButtonDown, exSpinDownButtonUp and exSpinDownButtonDown to change the visual appearance for the spin control. Use the exSliderRange and exSliderThumb to change the visual appearance for the slider control.
CalculatorType	21	Adds a drop down calculator to a node. Use the exCalcExecuteKeys, exCalcCannotDivideByZero, exCalcButtonWidth, exCalcButtonHeight, exCalcButtons, exCalcPictureUp, exCalcPictureDown to specify different options for calculator editor.

CloneType	268435456	The CloneType flag specifies that the current column uses the editor of a different column. The Column.Editor.EditType property must be CloneType + Index, where Index is the index of the column whose editor is used instead, in the current column. For instance, you have more columns that displays same data, and so you can use the same drop down for it, to select a different value. In other words, you define the editor once, and uses it on any other columns. For instance, Column.Editor.EditType = CloneType + 2, indicates that the Column uses the editor of the column with the index 2.

All editors support the following options:

- exLeftArrow, Disables focusing a new cell if the user presses the left arrow key while editing.
- exRightArrow, Disables focusing a new cell if the user presses the right arrow key while editing.
- exUpArrow, Disable focusing a new cell if the user presses the up arrow key while editing.
- exDownArrow, Disable focusing a new cell if the user presses the down arrow key while editing.
- exHomeKey, Disable focusing a new cell if the user presses the home key while editing.
- exEndKey, Disables focusing a new cell if the user presses the end key while editing.
- exPageUpKey, Disable focusing a new cell if the user presses the page up key while editing.
- exKeepSelBackColor. Keeps the selection background color while editor is visible.

constants exClipboardFormatEnum

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

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

constants exOLEDragOverEnum

State transition constants for the OLEDragOver event.

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

constants exOLEDropEffectEnum

Drop effect constants for OLE drag and drop events.

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

constants exOLEDropModeEnum

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

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

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

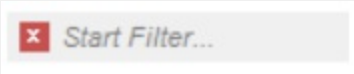
constants ExpandButtonEnum

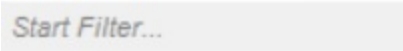
Defines how the control displays the expanding/collapsing buttons.

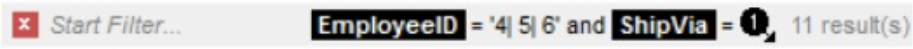
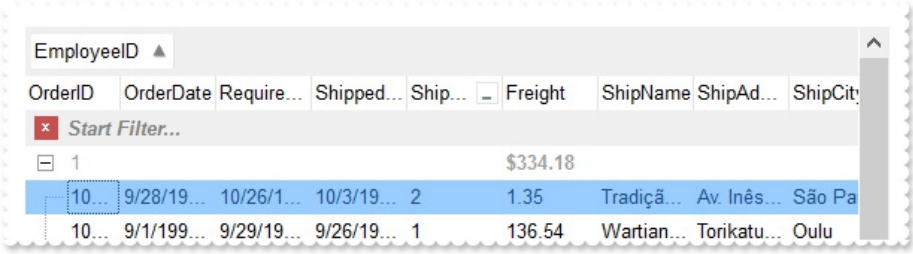
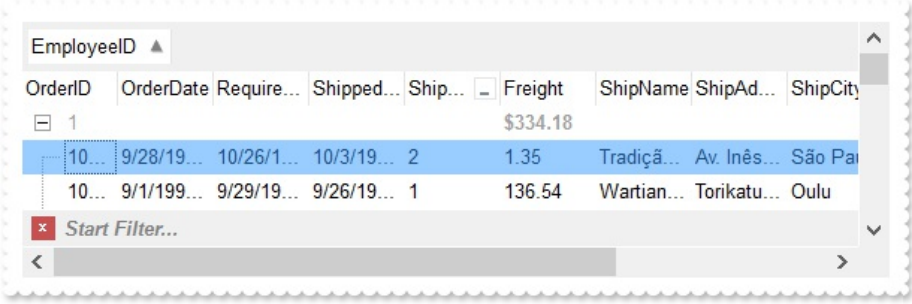
Name	Value	Description
exNoButtons	0	The control displays no expand buttons.
exPlus	-1	A plus sign is displayed for collapsed items, and a minus sign for expanded items.(⊞ ⊟)
exArrow	1	The control uses icons to display the expand buttons.(▸ ▾)
exCircle	2	The control uses icons to display the expand buttons. (⊕ ⊖)
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:

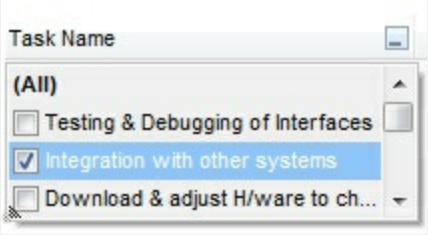
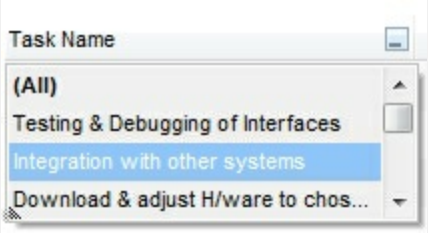
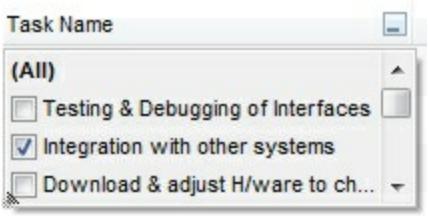
Name	Value	Description
exItemsWithoutChilds	0	Items (and parent-items) that match the filter are shown (no child-items are included)
exItemsWithChilds	1	Items (parent and child-items) that match the filter are shown
exRootsWithoutChilds	2	Only root-items (excludes child-items) that match the filter are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

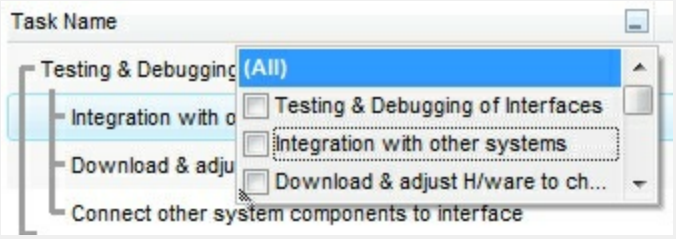
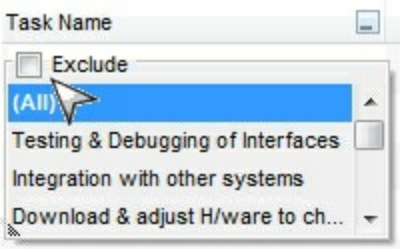
constants FilterListEnum

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,exFilterBarPatternToolt ...) properties defines the filter's tooltips.
exShowExclude	8192	This flag indicates whether the Exclude option is shown in the drop down filter window. This option has effect also if the drop down filter window shows a calendar control. The following screen shot shows the Exclude field in the drop down filter window: 
exShowBlanks	16384	This flag indicates whether the (Blanks) and (NonBlanks) items are shown in the filter's list

constants FilterPromptEnum

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

The filter prompt feature supports the following values:

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

- ' ' space delimits the patterns inside the filter

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

constants FilterTypeEnum

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

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

		and less than 100. 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.
exCheck	6	Only checked or unchecked items are included. The CellState property indicates the state of the cell's checkbox. The control filters for checked items, if the Filter property is "1". The control filters for unchecked items, if the Filter property is "0". A checked item has the the CellState property different than zero. An unchecked item has the CellState property on zero.
exImage	10	Filters items by icons. The CellImage property indicates the cell's icon.
exFilter	240	Only the items that are in the Filter property are included. The CellCaption property indicates the cell's caption.
exFilterDoCaseSensitive	256	By default, the filtering is case-insensitive. If this flag is set, the filtering is case-sensitive. This option can be combined with exFilter or exPattern flag to perform a case-sensitive filtering. For instance, the exFilter + exFilterDoCaseSensitive indicates that the column includes only the values that match exactly the values in the Filter property.
exFilterExclude	512	exFilterExclude

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	The control's gridlines are drawn using a geometric pen. The exGridLinesGeometric flag can be combined with any other flag. A geometric pen can have any width and can have any of the attributes of a brush, such as dithers and patterns. A cosmetic pen can only be a single pixel wide and must be a solid color, but cosmetic pens are generally faster than geometric pens. The width of a geometric pen is always specified in world units. The width of a cosmetic pen is always 1.

constants HierarchyLineEnum

Defines how the control paints the hierarchy lines.

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

constants HitTestInfoEnum

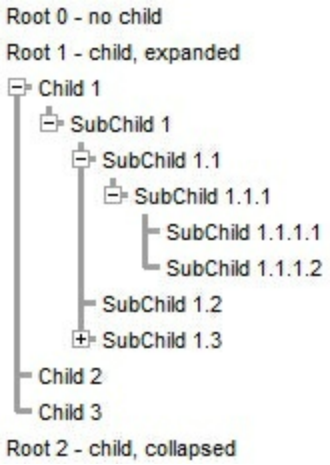
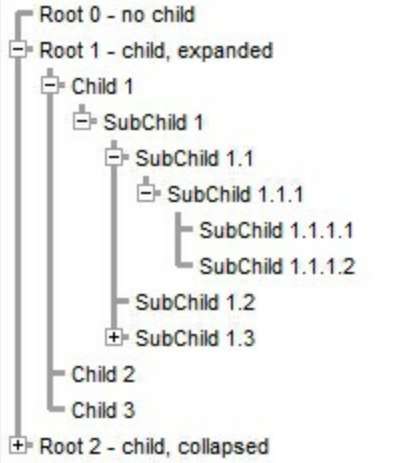
The HitTestInfoEnum expression defines the hit area within a cell. Use the ItemFromPoint property to determine the hit test code within the cell.

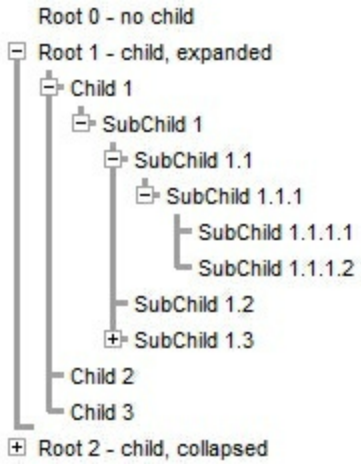
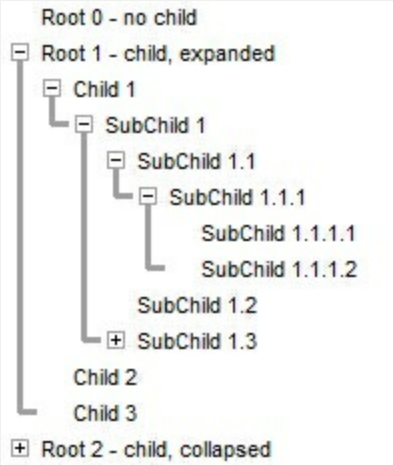
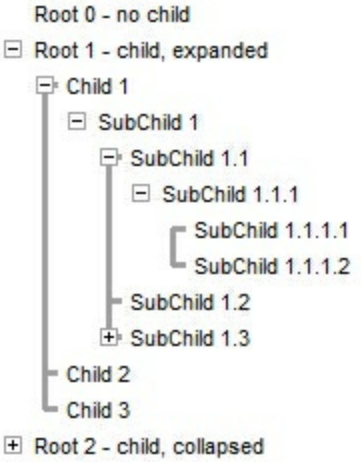
Name	Value	Description
exHTCell	0	In the cell's client area.
exHTExpandButton	1	In the +/- button associated with a cell. The HasButtons property specifies whether the cell displays a +/- sign to let user expands the item.
exHTCellIndent	2	In the indentation associated with a cell. The Indent property retrieves or sets the amount, in pixels, that child items are indented relative to their parent items.
exHTCellInside	4	On the icon, picture, check or caption associated with a cell.
exHTCellCaption	20	(HEXA 14) In the caption associated with a cell. The CellValue property specifies the cell's value.
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 CellValueFormat 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	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

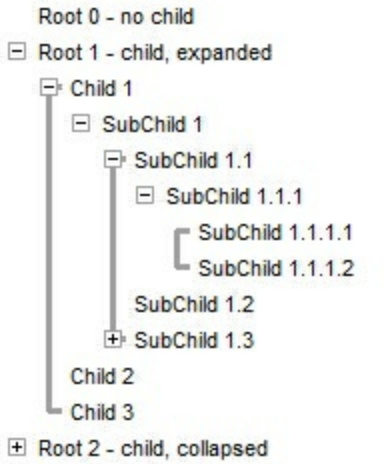
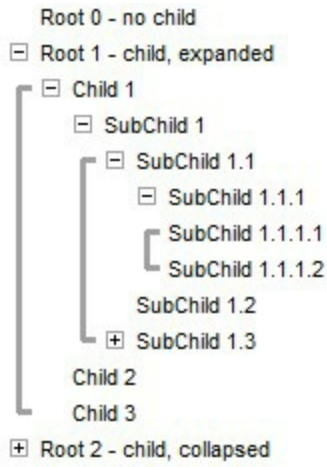
constants LinesAtRootEnum

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 InplaceAppearanceEnum

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

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

constants NumericEnum

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

Name	Value	Description
exInteger	-1	Allows editing numbers of integer type. The format of the integer number is: **[+/-]digit**, where **digit** is any combination of digit characters. This flag can be combined with exDisablePlus, exDisableMinus or exDisableSigns flags. For instance, the 0x3FF (hexa representation, 1023 decimal) value indicates an integer value with no +/- signs.
exAllChars	0	Allows all characters. No filtering.
exFloat	1	Allows editing floating point numbers. The format of the floating point number is: **[+/-]digit[.digit[[e/E/d/D][+/-]digit]]**, where digit is any combination of **digit** characters. Use the exEditDecimalSymbol option to assign a new symbol for '**.**' character (decimal values). This flag can be combined with exDisablePlus, exDisableMinus or exDisableSigns flags.
exFloatInteger	2	Allows editing floating point numbers without exponent characters such as e/E/d/D, so the accepted format is **[+/-]digit[.digit]**. Use the exEditDecimalSymbol option to assign a new symbol for '**.**' character (decimal values). This flag can be combined with exDisablePlus, exDisableMinus or exDisableSigns flags.
exDisablePlus	256	Prevents using the + sign when editing numbers. If this flag is included, the user can not add any + sign in front of the number.
exDisableMinus	512	Prevents using the - sign when editing numbers. If this flag is included, the user can not add any - sign in front of the number.
exDisableSigns	768	Prevents using the +/- signs when editing numbers. If this flag is included, the user can not add any +/- sign in front of the number. For instance exFloatInteger + exDisableSigns allows editing floating points numbers without using the exponent and plus/minus characters, so the allowed format is

digit[.digit]

constants PictureDisplayEnum

Specifies how a picture object is displayed.

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 ReadOnlyEnum

The ReadOnly property makes the control read-only. Use the Enabled property to disable the control. Use the Locked property to lock a specific editor. Use the CellEditorVisible property to hide the cell's editor.

Name	Value	Description
exReadWrite	0	(boolean False) The control allows changes. The user can use the cell's editor to change the cell's value.
exReadOnly	-1	(boolean True) The control is read only and the cell's editor is not visible.
exLocked	1	The control is read only, and the cell's editor is visible but locked. For instance, if the cell's editor contains a drop down portion, the user can display the drop down portion of the control, but it can't select a new value. Also, if the editor contains multiple buttons they are active as the control is not read only.

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. The ScrollBars property of the control specifies the scroll bars being visible in the control. By default, the ScrollBars property is exBoth, which indicates that both scroll bars of the component are being displayed only when they require.

- The horizontal scroll bar is not shown, if the ColumnAutoResize property is True, or if the ScrollBars property is exNoScroll. The horizontal scroll bar is shown if required, if the ScrollBars property is exBoth or exHorizontal, else it is always shown if the ScrollBars property is exDisableBoth or exDisableNoHorizontal
- The vertical scroll bar of the control is shown if required, if the ScrollBars is exBoth or exVertical, else if it is always shown if the ScrollBars property is exDisableBoth or exDisableVertical. For instance, if the ScrollBars property is exBoth OR exVScrollOnThumbRelease, the control's content is scrolled when the user releases the vertical thumb. If your data displays items with different heights, you should set the ScrollBySingleLine property on True.

Use the Scroll method to programmatically scroll the control's content to specified position. The ScrollPos property determines the position of the control's scroll bars. The ScrollWidth property specifies the width in pixels, of the vertical scroll bar. The ScrollHeight property specifies the height in pixels of the horizontal scroll bar. The ScrollOrderParts property specifies the order to display the parts of the scroll bar (buttons, thumbs and so on). The ScrollPartCaption property specifies the caption to be shown on any part of the scroll bar. Use the SelectPos property to select items giving its position.

The ScrollBars property supports a bitwise OR combination of the following values:

Name	Value	Description
exNoScroll	0	No scroll bars are shown. This flag should not be combined with any other.
exHorizontal	1	Only horizontal scroll bars are shown. This flag can be combined with any other flag greater or equal with 256.
exVertical	2	Only vertical scroll bars are shown. This flag can be combined with any other flag greater or equal with 256.
exBoth	3	Both horizontal and vertical scroll bars are shown. This flag can be combined with any other flag greater or equal with 256.
exDisableNoHorizontal	5	The horizontal scroll bar is always shown, it is disabled if it is unnecessary. This flag can be

		combined with any other flag greater or equal with 256.
exDisableNoVertical	10	The vertical scroll bar is always shown, it is disabled if it is unnecessary. This flag can be combined with any other flag greater or equal with 256.
exDisableBoth	15	Both horizontal and vertical scroll bars are always shown, disabled if they are unnecessary. This flag can be combined with any other flag greater or equal with 256.
exHScrollOnThumbRelease	256	Scrolls the control's content when the user releases the thumb of the horizontal scroll bar. Use this option to specify that the user scrolls the control's content when the thumb of the scroll box is released.
exVScrollOnThumbRelease	512	Scrolls the control's content when the user releases the thumb of the vertical scroll bar. Use this option to specify that the user scrolls the control's content when the thumb of the scroll box is released.
exHScrollEmptySpace	1024	Allows empty space, when control's content is horizontally scrolled to the end. If this flag is set, the last visible column, is displayed on leftmost position of the control, when the user horizontally scrolls to the end.
exVScrollEmptySpace	2048	Allows empty space, when control's content is vertically scrolled to the end. If this flag is set, the last visible item, is displayed on top of the control, when the user vertically scrolls to the end.

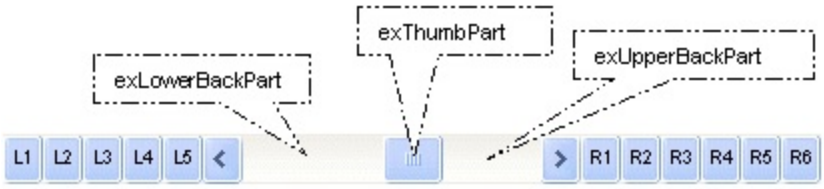
constants ScrollEnum

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 has been clicked.

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

constants SortOrderEnum

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

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

constants SortTypeEnum

The SortTypeEnum enumeration defines the types of sorting in the control. Use the SortType property to specifies the type of column's sorting.

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 column gets sorted numerical using the CellData property.
SortCellData	6	The column gets sorted numerical using the CellSortData property.
SortCellDataString	7	The CellSortData property indicates the values being sorted. The values are sorted as string.
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
exSliderVisualTheme	16	exSliderVisualTheme
exSpinVisualTheme	32	exSpinVisualTheme
exCheckBoxVisualTheme	64	exCheckBoxVisualTheme
exProgressVisualTheme	128	exProgressVisualTheme
exCalculatorVisualTheme	256	exCalculatorVisualTheme

constants ValidateValueType

The ValidateValueType specifies the type of validation that control supports. The CauseValidateValue property specifies whether the ValidateValue event is fired before Change event, so the user can validate the values being entered. The ValidateValue event is not fired if the CauseValidateValue property is False ~ exNoValidate. The ValidateValue event is fired once the user tries to leaves the focused cell (exValidateCell) or focused item (exValidateItem). The ValidateValueType enumeration supports the following values:

Name	Value	Description
exValidateCell	-1	The ValidateValue event is called just before leaving the cell. Use this option to validate the values per cell.
exNoValidate	0	The ValidateValue event is not fired.
exValidateItem	1	The ValidateValue event is fired when the user leaves the focused item. Use this option to validate the values per item.

constants VAlignmentEnum

Specifies the source's vertical alignment.

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

constants ValueFormatEnum

Defines how the cell's value is shown. The CellValueFormat property indicates the way the cell displays its content. The Def(exCellValueFormat) property indicates the format for all cells within the column. The CellValue property indicates the cell's value, content or formula. The ComputedField property indicates the formula to compute all cells in the column. The FormatColumn property indicates the format to be applied for cells in the columns.The ValueFormatEnum type supports can be a combination of the following values:

Name	Value	Description
exText	0	Standard text. No HTML tags are displayed
		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.

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

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

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

- **<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

Indicates a computed field. The CellValue property indicates the formula to compute the field. A computed field can display its content using the values from any other cell in the same item/row. For instance %1 + %2 indicates that the cell displays the addition from the second and third cells in the same item (cells are 0 based). For instance, if the

exComputedField	2	cells are of numeric format the result is the sum of two values, while if any of the cell is of string type it performs a concatenation of the specified cells. The ComputedField property indicates the formula to compute all cells in the column. The exComputedField can be combined with exText or exHTML. For instance, the exComputedField + exHTML indicates that the computed field may display HTML tags. The syntax for the CellValue property should be: **formula** where %n indicates the cell from the n-index. The operation being supported are listed bellow. For instance %1 + %2 indicates the sum of all cells in the second and third column from the current item.
		Indicates a total/subtotal field. The CellValue property indicates the formula for total field that includes an aggregate function such as: sum, min, max, count, avg. The exTotalField can be combined with exText or exHTML. For instance, the exTotalField + exHTML indicates that the total field may display HTML tags. The syntax for the CellValue property should be: **aggregate**(**list**,**direction**,**formula**) where: **aggregate** must be one of the following: • *sum* - calculates the sum of values. • *min* - retrieves the minimum value. • *max* - retrieves the maximum value. • *count* - counts the number of items. • *avg* - calculates the average of values. **list** must be one of the following: • a *long* expression that specifies the index of the item being referred. • a predefined string expression as follows: ◦ *all* - indicates all items, so the formula is

		being applied to all items. The direction has no effect. ◦ *current* - the current item. ◦ *parent* - the parent item. ◦ *root* - the root item. **direction** must be one of the following: • *dir* - collects the direct descendents. • *rec* - collects the leaf descendents (leaf items). A leaf item is an item with no child items. • *all* - collects all descendents. Currently, the following items are excluded by aggregate functions:
exTotalField	4	• *not-sortable items*. The SortableItem property specifies whether the item can be sorted (a sortable item can change its position after sorting, while a not-sortable item keeps its position after sorting. • *not-selectable items*. The SelectableItem property specifies whether the user can selects/focus the specified item. • *divider items*. The ItemDivider property specifies whether the item displays a single cell, instead displaying whole cells. In conclusion, aggregate functions counts ONLY items that are: • *sortable*, SortableItem is True, by default. • *selectable*, SelectableItem is True, by default. • *not divider*, ItemDivider is -1, by default. Shortly, by setting to a different value to any of these properties, makes the item to be ignored by the aggregate functions. For instance • count(current,dir,1) counts the number of child items (not implies recursively child items). • count(current,all,1) counts the number of all

child items (implies recursively child items).
- count(current,rec,1) counts the number of leaf items (implies recursively leaf items).
- count(current,rec,1) counts the number of leaf items (a leaf item is an item with no child items).
- sum(parent,dir,%1=0?0:1) counts the not-zero values in the second column (%1)
- sum(parent,dir,%1 + %2) indicates the sum of all cells in the second (%1) and third (%2) column that are directly descendent from the parent item.
- sum(all,rec,%1 + %2) sums all leaf cells in the second (%1) and third (%2) columns.

The **formula** on the CellValue property (if the CellValueFormat property indicates the exComputedField or exTotalField) may include the formatting operators as follows:

The expression supports cell's identifiers as follows:

- ***%0**, **%1**, **%2**, ... specifies the value of the cell in the column with the index 0, 1 2, ... The CellValue property specifies the cell's value. For instance, "%0 format ``" formats the value on the cell with the index 0, using current regional setting, while "int(%1)" converts the value of the column with the index 1, to integer.*
- ***%C0**, **%C1**, **%C2**, ... specifies the caption of the cell, or the string the cell displays in the column with the index 0, 1 2, ... The CellCaption property specifies the cell's caption. The cell's value may be different than what the cell displays as a string. For instance, let's say a cell display HTML format. The %0 returns the html format including the HTML tags, while %C0 returns the cell's content as string without HTML tags. For instance, "upper(%C1)" converts the caption of the cell with the index 1, to upper case, while "%C0 left 2" returns the leftmost two characters on the cell with the index 0.*
- ***%CD0**, **%CD1**, **%CD2**, ... 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 instance, "%CD0 = `your user data`" specifies all cells whose CellData property is `your user data`, on the column with the index 0.*
- ***%CS0**, **%CS1**, **%CS2**, ... specifies the cell's state in the column with the index 0, 1 2, ... The CellState property specifies the cell's state, and so it indicates whether the cell is checked or un-checked. For instance, "%CS0" defines all checked items on the column with the index 0, or "not %CS1" defines all un-checked items in the column with the index 1.*

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

Usage examples:

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 indicates 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.

constants ViewModeEnum

The ViewModeEnum type specifies the ways the control may display the data. The ViewMode property specifies the way the control is displaying data. The ViewModeOption property specifies options for given view modes.

Name	Value	Description
exTableView	0	Shows the items as rows in a table.
exCardView	1	Displays each record's information as fields on a card.

constants ViewModeOptionEnum

The ViewModeOptionEnum type indicates the set of options user can access for different types of views. The ViewMode property specifies the way the control displays its data. The ViewModeOption property specifies the option for a given view. The options that start with exCardView have effect only if the ViewMode property is exCardView. The options that start with exTableView have effect only if the ViewMode property is exTableView. All other options are valid for all modes.

Name	Value	Description
exBorderWidth	0	Specifies the width in pixels of the empty border inside the view. The option is valid for all view modes.
exBorderHeight	1	Specifies the height in pixels of the empty border inside the view. The option is valid for all view modes.
exCardViewWidth	2	Specifies the width in pixels of the card. If this option is 0, the width of the card is the same with the control's visible area as follows, if the ColumnAutoResize property is True, the width of the card is the same as the control's client area. If the ColumnAutoResize property is False, the width of the card is the sum of the width of all visible columns. If this option is 0, the user can't resize the cards at runtime, and so the exCardViewVResizeLine has no effect. (long expression, 128)
exCardViewHeight	3	Specifies the height in pixels of the card, not including the height of the card's title. The height of the card's title is determined by the DefaultItemHeight property. Use the exCardViewTitleFormat option on empty string to hide the titles of the cards. If the exCardViewHeight option is 0, the height of the card is computed as follows, if the exCardViewColumns option is 0, the height of the card is the same as the height of the control's client area, else the height of the card is the result of division the height of the control's client area by exCardViewColumns option. If this option is 0, the user can't resize the card at runtime, and so the exCardViewHResizeLine option has no effect. (long expression, 144)

exCardViewFormat	4	Specifies the arrangement of the fields in the cards. The exCardViewFormat supports CRD format. (string expression, "1/2/3/4/5/6/7")
exCardViewTitleFormat	5	Specifies the arrangement of the fields in the title of the card. The exCardViewTitleFormat supports CRD format. If the exCardViewTitleFormat option is empty, the cards are displayed without a title. If the card is collapsed, the card displays only its title. Use the ExpandCard property to expand or collapse programmatically a card. The HasButtons property indicates whether the control displays an expand/collapse button in the title of the card. (string expression, "0")
exCardViewTitleBackColor	6	Specifies the visual appearance of the title of the card. As all color properties, it supports displaying a skin object as well. (color expression, 0)
exCardViewTitleForeColor	7	Specifies the foreground color for cells in the title of the card. (color expression, 0)
exCardViewBackColor	8	Specifies the visual appearance of the card without the title. As all color properties, it supports displaying a skin object as well. (color expression, 0)
exCardViewBorderWidth	9	Specifies the width in pixels of the empty border between cards. Specifies the distance in pixels between two cards. (long expression, 4)
exCardViewBorderHeight	10	Specifies the height in pixels of the empty border between cards. Specifies the distance in pixels between two cards. (long expression, 4)
exCardViewLeftToRight	11	Retrieves or sets a value that indicates whether the cards are arranged from left to right or from top to right. If the exCardViewLeftToRight option is True, the exCardViewColumns indicates the number of columns of cards being displayed. If the exCardViewLeftToRight option is False, the exCardViewColumns indicates the number of rows of cards being displayed. (boolean expression, True)
		Specifies the number of columns of cards being displayed as follows if the exCardViewLeftToRight option is True, the exCardViewColumns indicates

exCardViewColumns	12	the number of columns of cards being displayed, If the exCardViewLeftToRight option is False, the exCardViewColumns indicates the number of rows of cards being displayed. (long expression, 0)
exCardViewTitleReadOnly	13	Specifies whether the title of the card is read only. (boolean expression, 4)
exCardViewVResizeLine	14	Gets or sets a value that indicates whether the control draws the vertical resizing lines. The resizing lines are not shown if the exCardViewWidth property is 0. Use the ItemsAllowSizing property to allow resizing the cards at run-time. (boolean expression, False)
exCardViewHResizeLine	15	Gets or sets a value that indicates whether the control draws the horizontal resizing lines. The resizing lines are not shown if the exCardViewHeight property is 0. Use the ItemsAllowSizing property to allow resizing the cards at run-time. (boolean expression, False)

Appearance object

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

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

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

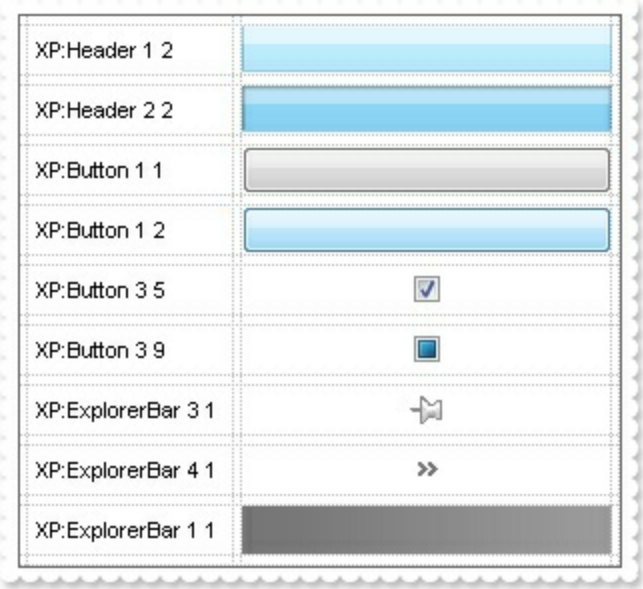
Adds or replaces a skin object to the control.

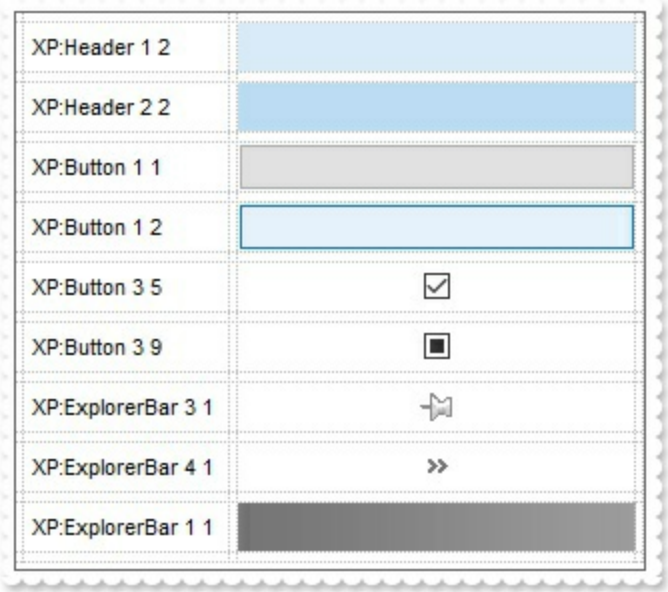
Type	Description
ID as Long	A Long expression that indicates the index of the skin being added or replaced. The value must be between 1 and 126, so Appearance collection should holds no more than 126 elements.
	The Skin parameter of the Add method can a STRING as explained bellow, a BYTE[] / safe arrays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

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

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

Skin as Variant

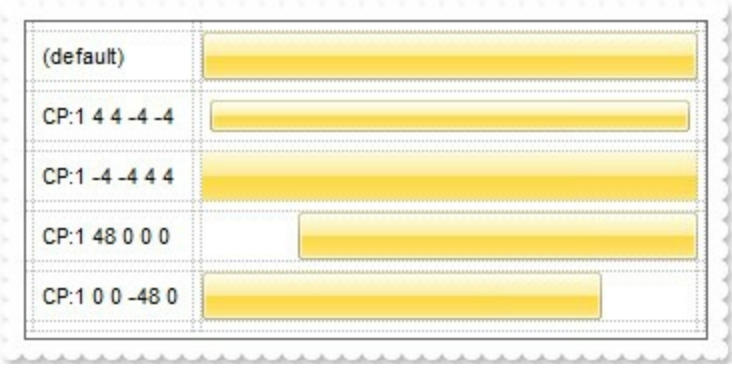




- A copy of another skin with different coordinates (position, size), if the Skin parameter starts with "**CP**:". Use this option, to display the EBN, using different coordinates (position, size). By default, the EBN skin object is rendered on the part's client area. Using this option, you can display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

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

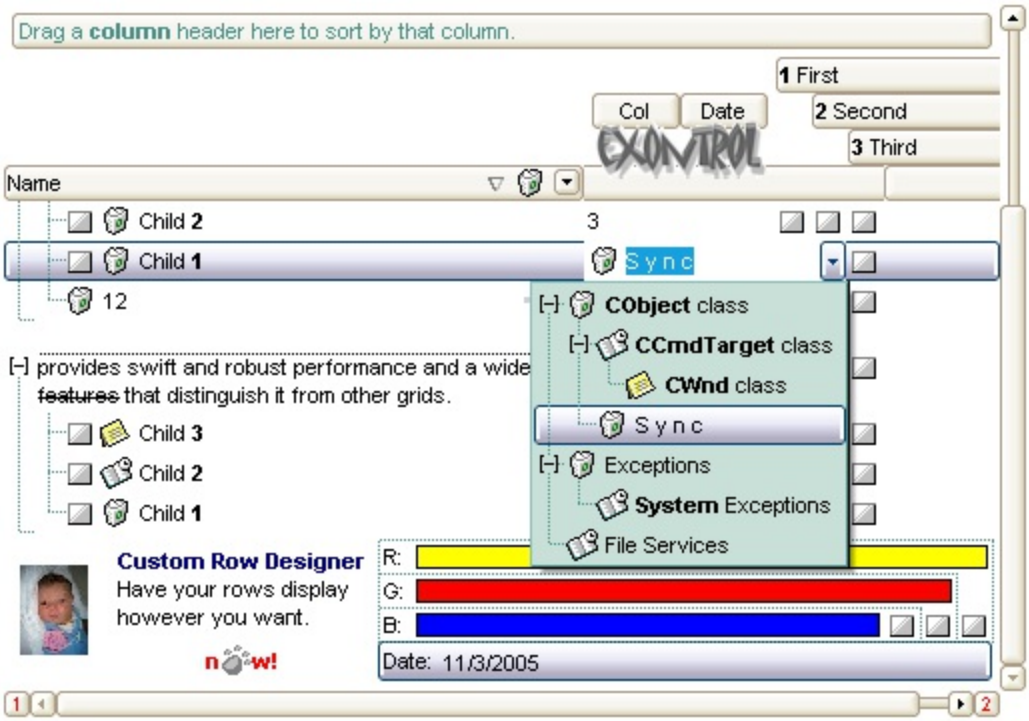
The following screen shot shows the same EBN being displayed, using different CP: options:



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

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

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



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

- control's border, Appearance property
- control's **header bar**, BackColorHeader property
- control's **filter bar**, FilterBarBackColor property
- control's **sort bar**, BackColorSort property
- the caption of the control's sort bar, BackColorSortCaption property
- **selected item** or cell, SelBackColor property

- **item**, ItemBackColor property
- **cell**, CellBackColor property
- CellImage, CellImages, HeaderImage, CheckImage or RadioImage, HasButtonsCustom property
- cell's **button**, "**drop down**" filter bar button, "close" filter bar button, tooltips, and so on, Background property

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

The following VB sample applies the "  " skin to the selected item(s):

```
With Grid1
   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 VB sample changes the visual appearance of the selected item, using a Windows XP part from the current theme:

```
With Grid1
   With .VisualAppearance
      .Add &H23, "XP:ScrollBar 2 1"
   End With
   .SelForeColor = RGB(0, 0, 0)
   .SelBackColor = &H23000000
End With
```

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

```
#include "Appearance.h"
m_grid.GetVisualAppearance().Add( 0x23,
COleVariant(_T("D:\\Temp\\ExGrid_Help\\selected.ebn")) );
```

```
m_grid.SetSelBackColor( m_grid.GetSelBackColor() | 0x23000000 );
m_grid.SetSelForeColor( 0 );
```

The following C++ sample change the visual appearance of the selected item(s), using a Windows XP part from the current theme:

```
#include "Appearance.h"
m_grid.GetVisualAppearance().Add( 0x23, COleVariant(_T("XP:ScrollBar 2 1")) );
m_grid.SetSelBackColor( 0x23000000 );
m_grid.SetSelForeColor( 0 );
```

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

```
With AxGrid1
   With .VisualAppearance
      .Add(&H23, "D:\Temp\ExGrid_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 VB.NET sample changes the visual appearance of the selected item, using a Windows XP part from the current theme:

```
With AxGrid1
   With .VisualAppearance
      .Add(&H23, "XP:ScrollBar 2 1")
   End With
   .SelForeColor = Color.Black
   .Template = "SelBackColor = 587202560"
End With
```

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

```
axGrid1.VisualAppearance.Add(0x23, "D:\\Temp\\ExGrid_Help\\selected.ebn");
axGrid1.Template = "SelBackColor = 587202560";
```

The following C# sample changes the visual appearance of the selected item, using a Windows XP part from the current theme:

```
axGrid1.VisualAppearance.Add(0x23, "XP:ScrollBar 2 1");
axGrid1.Template = "SelBackColor = 587202560";
```

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

```
With thisform.Grid1
   With .VisualAppearance
      .Add(35, "D:\Temp\ExGrid_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 following VFP sample changes the visual appearance of the selected item, using a Windows XP part from the current theme:

```
With thisform.Grid1
   With .VisualAppearance
      .Add(35, "XP:ScrollBar 2 1")
   EndWith
   .SelForeColor = RGB(0, 0, 0)
   .SelBackColor = 587202560
EndWith
```

The first screen 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
   CellValueFormat(h,2) = 1
   h1 = InsertItem(h,,16)
   CellValue(h1,1) = 17
   h1 = InsertItem(h,,2)
   CellValue(h1,1) = 11
   h1 = InsertItem(h,,2)
   CellValue(h1,1) = 9
   ExpandItem(h) = True
   h = InsertItem(,,"Group 2")
   CellEditorVisible(h,0) = False
   CellEditorVisible(h,1) = False
   CellValueFormat(h,2) = 1
   h1 = InsertItem(h,,16)
   CellValue(h1,1) = 9
   h1 = InsertItem(h,,12)
   CellValue(h1,1) = 11
   h1 = InsertItem(h,,2)
   CellValue(h1,1) = 2
   ExpandItem(h) = True
   SelectItem(h) = True
}
EndUpdate
```

The second screen shot was generated using the following template:

```
BeginUpdate
Images("gBJJgBAIFAAJAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaGEaAIAEEbjMjIErlktl0v



VisualAppearance
{
   ' Header
```

```
Add(1,"gBFLBCJwBAEHhEJAEGg4BawDg6AADACAxRDAMgBQKAAzQFAYZhoHKGAAGEYxR

    ' HeaderFilterBarButton

Add(2,"gBFLBCJwBAEHhEJAEGg4BAQEg6AADACAxRDAMgBQKAAzQFAYZhoHKGAAGEYxR



Add(3,"gBFLBCJwBAEHhEJAEGg4BBAEg6AADACAxRDAMgBQKAAzQFAYZhoHKGAAGEYxRg

    ' SelectedItem
    Add(4,
"gBFLBCJwBAEHhEJAEGg4BV4Fg6AABACAxWgKBADQKAAyDIKsEQGGIZRhhGIwAgaFIXQK



Add(5,"gBFLBCJwBAEHhEJAEGg4Ba4Fg6AABACAxWgKBADQKAAyDIKsEQGGIZRhhGIwAga



Add(6,"gBFLBCJwBAEHhEJAEGg4BaAFg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUA



Add(7,"gBFLBCJwBAEHhEJAEGg4BYIEg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUAI



    ' DropDownButton

Add(14,"gBFLBCJwBAEHhEJAEGg4BbYFg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhU



Add(15,"gBFLBCJwBAEHhEJAEGg4Bf4Fg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUA



}
BackColorHeader = 16777216          '0x01BBGGRR
BackColorSortBarCaption = 33488896     '0x01BBGGRR
FilterBarBackColor = 16777216         '0x01BBGGRR
Background(0) = 33554432              '0x02BBGGRR
```

```
Background(1) = 50331648            '0x03BBGGRR
Background(2) = 67108864            '0x04BBGGRR
Background(3) = 83886080            '0x05BBGGRR
Background(4) = 234881024           '0x0EBBGGRR
Background(5) = 251658240           '0x0FBBGGRR
Background(6) = 83886080            '0x05BBGGRR
Background(7) = 67108864            '0x04BBGGRR
Background(8) = 67108864            '0x04BBGGRR
Background(9) = 67108864            '0x04BBGGRR
Background(10) = 100663296            '0x06BBGGRR
Background(11) = 100663296            '0x06BBGGRR
Background(12) = 100663296            '0x06BBGGRR
Background(13) = 100663296            '0x06BBGGRR
Background(14) = 100663296            '0x06BBGGRR
BackGround(15) = RGB(208,207,224)
BackGround(16) = 67108864
BackGround(17) = RGB(216, 215, 232)

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

MarkTooltipCells = True
MarkSearchColumn = False
Indent = 15
LinesAtRoot = 1
HasButtons = 4
HasButtonsCustom(0) = 4
HasButtonsCustom(1) = 5
SortBarVisible = True
DefaultItemHeight = 20
HeaderHeight = 20
SortBarHeight = 20
```

```
BackColor = RGB(255,255,255)
BackColorLevelHeader = RGB(255,255,255)
DrawGridLines = -1
ScrollBySingleLine = True
ShowFocusRect = False

Columns
{
   "Name"
   {
      DisplayFilterButton = True
      DisplayFilterDate = True
      Width = 96
      AutoSearch = 1
      HeaderImage = 1
      HeaderImageAlignment = 2
   }
   "Value"
   {
      HeaderBold = True
      Editor
      {
         EditType = 3
         AddItem(1,"1. First",1)
         AddItem(2,"2. Second",2)
         AddItem(3,"3. Third",3)
      }
   }
   1
   {
      AllowSizing = False
      HTMLCaption = "1 First"
      Def(0) = True
      LevelKey = 1
      Width = 18
      PartialCheck = True
   }
```

```
   2
   {
      AllowSizing = False
      HTMLCaption = "2 Second"
      Def(0) = True
      LevelKey = 1
      Width = 18
      PartialCheck = True
   }
   3
   {
      AllowSizing = False
      HTMLCaption = "3 Third"
      Def(0) = True
      LevelKey = 1
      Width = 18
      PartialCheck = True
   }
   ""
   {
      LevelKey = 1
      Width = 20
   }
}
Items
{
   Dim h, h1,hx
   h = AddItem("exGridprovides swift and robust performance and a wide range of
formatting features that distinguish it from other grids.")
   CellSingleLine(h,0) = False
   CellValueFormat(h,0) = 1
   CellEditorVisible(h,1) = False
   CellToolTip(h,0) = ""
   CellMerge(h,0) = 1

   h1 = InsertItem(h,,"Child 1")
   CellValueFormat(h1,0) = 1
```

```
   CellHasCheckBox(h1,0) = True
   CellValue(h1,1) = 1
   CellToolTip(h1,0) = "exGrid
provides swift and robust performance and a wide range of formatting features that
distinguish it from other grids"
   CellImage(h1,0) = 1

   h1 = InsertItem(h,,"Child 2")
   CellValueFormat(h1,0) = 1
   CellHasCheckBox(h1,0) = True
   CellValue(h1,1) = 2
   CellForeColor(h1,1)= RGB(0,0,255)
   CellImage(h1,0) = 2

   h1 = InsertItem(h,,"Child 3")
   CellValueFormat(h1,0) = 1
   CellHasCheckBox(h1,0) = True
   CellValue(h1,1) = 3
   CellImage(h1,0) = 3
   CellState(h1,3) = 1

'  ExpandItem(h) = True

   h = AddItem("")
   ItemDivider(h) = 1
   SelectableItem(h) = False
   ItemHeight(h)= 38
   CellPicture(h,1) =
"gBHJJGHA5MIgAEIe4AAAFAoEDQXCoaEIdEkVi4lEgqEovElVF8cF40F0jGw5FQdHI0EsrF0rk4

   CellWidth(h,1) = 42
   CellValue(h,1) = "Not selectable item"
   CellEditorVisible(h,1) = False
   CellValueFormat(h,1) = 1
   CellHAlignment(h,1) = 1

   h1 = SplitCell(h2,1)
```

```
CellWidth(,h1) = 18
CellHasCheckBox(,h1) = True
CellEditorVisible(,h1) = False

h = AddItem("Root 2")
CellEditorVisible(h,1) = False
ItemBold(h) = True
CellMerge(h,0) = 1
CellMerge(h,0) = 2
CellEditor(h,0)
{
   EditType = 1
}

h1 = InsertItem(h,,"Child 1")
CellValueFormat(h1,0) = 1
CellValueFormat(h1,1) = 1
CellHasCheckBox(h1,0) = True
CellValue(h1,1) = 3
CellImage(h1,0) = 1
CellEditor(h1,1)
{
   EditType = 3
   DropDownAutoWidth = False
   AddItem(1,"CObject class", 1 )
   InsertItem(2,"CCmdTarget class", 2, 1 )
   InsertItem(3,"CWnd class", 3, 2 )
   InsertItem(6,"S y n c", 1, 1 )
   AddItem(4,"Exceptions", 1 )
   InsertItem(7,"System Exceptions", 2,4 )
   AddItem(5,"File Services", 2 )
   ExpandAll
   ItemToolTip(1) = "CObject tooltip
You can assign a tooltip to a predefined value in a drop down list editor. Multiple-lines,
HTML format supported."
   Option(37) = 1
}
```

```
CellMerge(h1,1) = 2
CellMerge(h1,1) = 3

h1 = InsertItem(h,,"Child 2")
CellValueFormat(h1,0) = 1
CellValueFormat(h1,1) = 1
CellHasCheckBox(h1,0) = True
CellValue(h1,1) = 3
CellImage(h1,0) = 1
CellEditor(h1,1)
{
   EditType = 21
   Option(105) =
"gBCJr2AwAg0HG0HFEHDEMg4BAENg4AYEFADEB0GjETjgAf8fV6dQB4JQukkmksnlUplkol0

   Option(106) =
"gBCJr2AwAg0HG0HFEHDEMg4BAENg4AYEFADEB0GjETjgAf8fV6dQB4JQukkmksnlUplkol0

}

h1 = InsertItem(h,,"Child 3")
CellValueFormat(h1,0) = 1
CellValueFormat(h1,1) = 1
CellHasCheckBox(h1,0) = True
CellValue(h1,1) = 7
CellImage(h1,0) = 2
CellEditor(h1,1)
{
   EditType = 6
   AddItem(1,"1. MN",1)
   AddItem(2,"2. FR",2)
   AddItem(4,"4. GD",3)
   AddItem(8,"8. BT",1)
}

h1 = InsertItem(h,,"Child 4")
CellValueFormat(h1,0) = 1
```

```
    CellHasCheckBox(h1,0) = True
    CellValue(h1,1) = 70
    CellImage(h1,0) = 3
    CellEditor(h1,1)
    {
        EditType = 13
    }

    h1 = InsertItem(h,,12)
    CellMerge(h1,0) = 1
    CellImage(h1,0) = 1
    CellValue(h1,1) = 34
    CellEditor(h1,0)
    {
        Numeric = -1
        EditType = 20
        Option(45) = True
        Option(41) = 120
    }
    ExpandItem(h) = True
}
EndUpdate
```

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

Control/ClassName	Part	States
BUTTON	BP_CHECKBOX = 3	CBS_UNCHECKED 1 CBS_UNCHECKE 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	
	EBP_HEADERCLOSE = 2	EBHC_NORMAL = EBHC_HOT = 2 EBHC_PRESSED =
	EBP_HEADERPIN = 3	EBHP_NORMAL = EBHP_HOT = 2 EBHP_PRESSED = EBHP_SELECTEDN 4 EBHP_SELECTE EBHP_SELECTEDP 6 EBM_NORMAL = 1

	EBP_IEBARMENU = 4	= 2 EBM_PRESSEI
	EBP_NORMALGROUPBACKGROUND = 5	
	EBP_NORMALGROUPCOLLAPSE = 6	EBNGC_NORMAL EBNGC_HOT = 2 EBNGC_PRESSED
	EBP_NORMALGROUPEXPAND = 7	EBNGE_NORMAL EBNGE_HOT = 2 EBNGE_PRESSED
	EBP_NORMALGROUPHEAD = 8	
	EBP_SPECIALGROUPBACKGROUND = 9	
	EBP_SPECIALGROUPCOLLAPSE = 10	EBSGC_NORMAL EBSGC_HOT = 2 EBSGC_PRESSED
	EBP_SPECIALGROUPEXPAND = 11	EBSGE_NORMAL EBSGE_HOT = 2 EBSGE_PRESSED
	EBP_SPECIALGROUPHEAD = 12	
HEADER	HP_HEADERITEM = 1	HIS_NORMAL = 1 2 HIS_PRESSED =
	HP_HEADERITEMLEFT = 2	HILS_NORMAL = 1 = 2 HILS_PRESSEI
	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	
	LVP_LISTITEM = 1	LIS_NORMAL = 1 2 LIS_SELECTED = 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	
	RP_CHEVRON = 4	CHEVS_NORMAL = CHEVS_HOT = 2

		CHEVS_PRESSED
	RP_CHEVRONVERT = 5	
	RP_GRIPPER = 1	
	RP_GRIPPERVERT = 2	
SCROLLBAR	SBP_ARROWBTN = 1	ABS_DOWNDISAB ABS_DOWNHOT, ABS_DOWNNORM ABS_DOWNPRESS ABS_UPDISABLED ABS_UPHOT, ABS_UPNORMAL, ABS_UPPRESSED, ABS_LEFTDISABLI ABS_LEFTHOT, ABS_LEFTNORMA ABS_LEFTPRESSE ABS_RIGHTDISAB ABS_RIGHTHOT, ABS_RIGHTNORM ABS_RIGHTPRESS
	SBP_GRIPPERHORZ = 8	
	SBP_GRIPPERVERT = 9	
	SBP_LOWERTRACKHORZ = 4	SCRBS_NORMAL = SCRBS_HOT = 2 SCRBS_PRESSED SCRBS_DISABLED
	SBP_LOWERTRACKVERT = 6	SCRBS_NORMAL = SCRBS_HOT = 2 SCRBS_PRESSED SCRBS_DISABLED
	SBP_THUMBBTNHORZ = 2	SCRBS_NORMAL = SCRBS_HOT = 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
	TABP_TOPTABITEMRIGHTEDGE = 7	TTIRES_NORMAL TTIRES_HOT = 2 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	
TOOLBAR	TP_BUTTON = 1	TS_NORMAL = 1 T TS_PRESSED = 3 TS_DISABLED = 4 TS_CHECKED = 5 TS_HOTCHECKED
	TP_DROPDOWNBUTTON = 2	TS_NORMAL = 1 T TS_PRESSED = 3 TS_DISABLED = 4 TS_CHECKED = 5 TS_HOTCHECKED
	TP_SPLITBUTTON = 3	TS_NORMAL = 1 T TS_PRESSED = 3 TS_DISABLED = 4 TS_CHECKED = 5 TS_HOTCHECKED
	TP_SPLITBUTTONDROPDOWN = 4	TS_NORMAL = 1 T TS_PRESSED = 3 TS_DISABLED = 4 TS_CHECKED = 5 TS_HOTCHECKED
	TP_SEPARATOR = 5	TS_NORMAL = 1 T TS_PRESSED = 3 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 = 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
	TKP_THUMBTOP = 5	TUTS_NORMAL = TUTS_HOT = 2 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 =

WP_MAX_BUTTON	HTS_DISABLED = 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 =
	FS_ACTIVE = 1 FS

WP_SMALLFRAMEBOTTOM = 12	= 2
WP_SMALLFRAMEBOTTOMSIZINGTEMPLATE = 37	
WP_SMALLFRAMELEFT = 10	FS_ACTIVE = 1 FS = 2
WP_SMALLFRAMELEFTSIZINGTEMPLATE = 33	
WP_SMALLFRAMERIGHT = 11	FS_ACTIVE = 1 FS = 2
WP_SMALLFRAMERIGHTSIZINGTEMPLATE = 35	
WP_SMALLHELPBUTTON	HBS_NORMAL = 1 = 2 HBS_PUSHED HBS_DISABLED =
WP_SMALLMAXBUTTON	MAXBS_NORMAL MAXBS_HOT = 2 MAXBS_PUSHED = MAXBS_DISABLED
WP_SMALLMAXCAPTION = 6	MXCS_ACTIVE = 1 MXCS_INACTIVE = MXCS_DISABLED
WP_SMALLMINCAPTION = 4	MNCS_ACTIVE = 1 MNCS_INACTIVE = MNCS_DISABLED
WP_SMALLRESTOREBUTTON	RBS_NORMAL = 1 = 2 RBS_PUSHED RBS_DISABLED =
WP_SMALLSYSBUTTON	SBS_NORMAL = 1 = 2 SBS_PUSHED SBS_DISABLED =
WP_SYSBUTTON = 13	SBS_NORMAL = 1 = 2 SBS_PUSHED SBS_DISABLED =
WP_VERTSCROLL = 27	VSS_NORMAL = 1 = 2 VSS_PUSHED VSS_DISABLED =
WP_VERTTHUMB = 28	VTS_NORMAL = 1 2 VTS_PUSHED = VTS_DISABLED =

method Appearance.Clear ()

Removes all skins in the control.

Type	Description

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

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

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

method Appearance.Remove (ID as Long)

Removes a specific skin from the control.

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

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

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

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

property Appearance.RenderType as Long

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

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

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

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

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

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

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

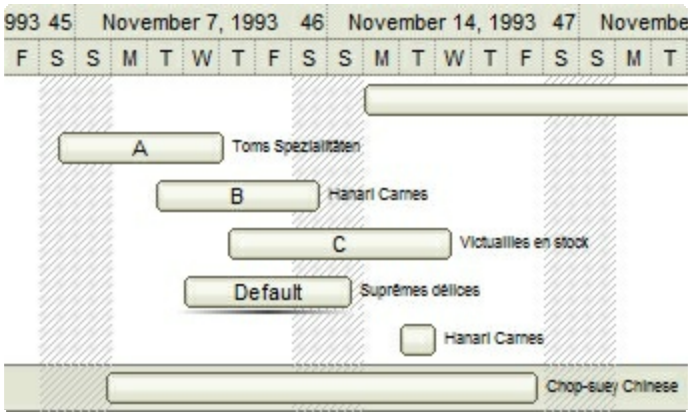
In the following screen shot the following objects displays the current EBN with a different color:

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

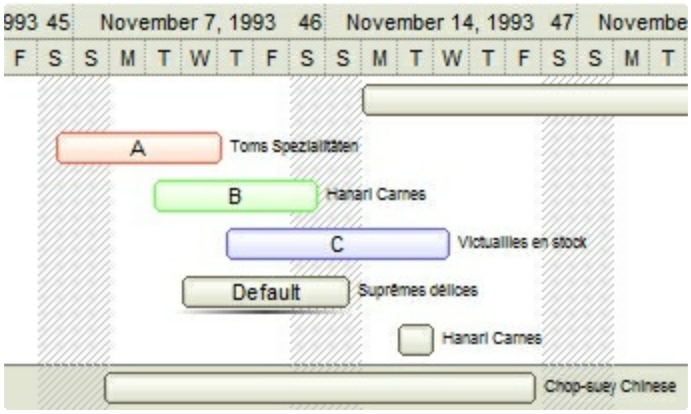
- "C" in Blue (RGB(0,0,255), for instance the bar's property exBarColor is 0x1FF0000
- "Default", no color is specified, for instance the bar's property exBarColor is 0x1000000

The RenderType property could be one of the following:

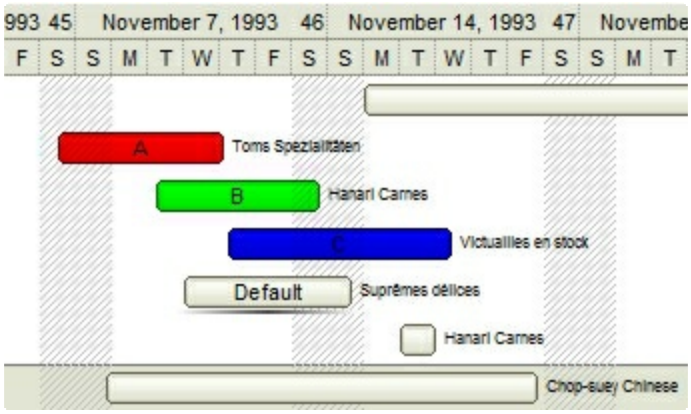
- ***-3**, no color is applied*. For instance, the BackColorHeader = &H1FF0000 is displayed as would be .BackColorHeader = &H1000000, so the 0xFF0000 color (Blue color) is ignored. You can use this option to allow the control displays the EBN colors or not.



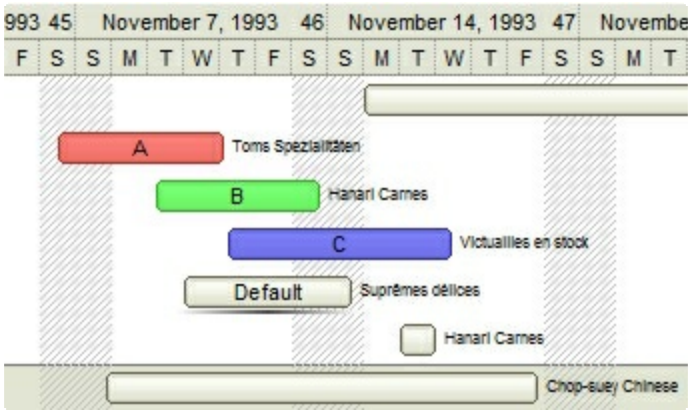
- ***-2**, OR-color scheme*. The color to be applied on the part of the control is a OR bit combination between the original EBN color and the specified color. For instance, the BackColorHeader = &H1FF0000, applies the OR bit for the entire Blue channel, or in other words, it applies a less Blue to the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



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

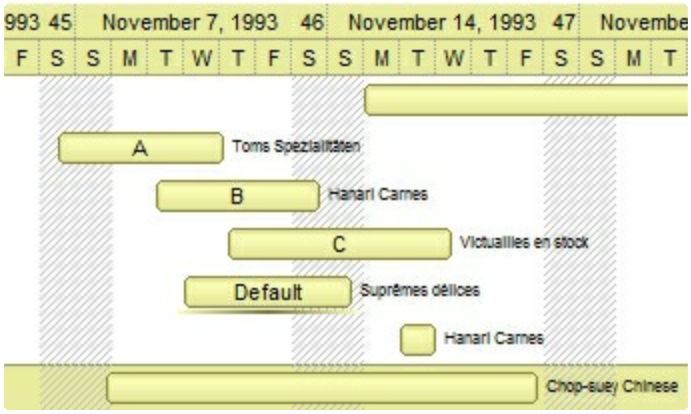


- ***0**, default*, the specified color is applied to the EBN. For instance, the BackColorHeader = &H1FF0000, applies a Blue color to the object. This option could be used to specify any color for the part of the components, that support EBN objects, not only solid colors.

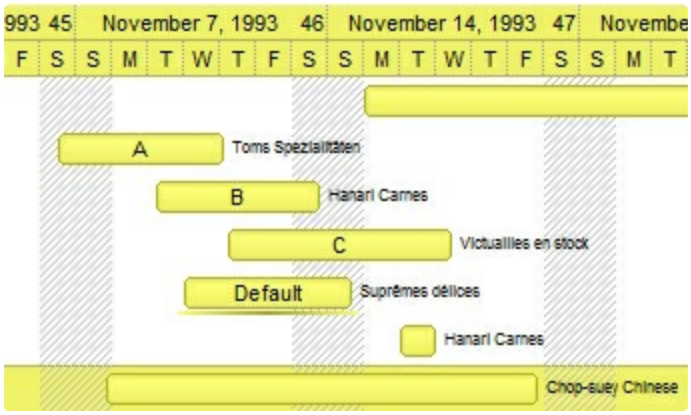


- **0xAABBGGRR**, where the AA a value between 0 to 255, which indicates the transparency, and RR, GG, BB the red, green and blue values. This option applies the same color to all parts that displays EBN objects, whit ignoring any specified color in the color property. For instance, the RenderType on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

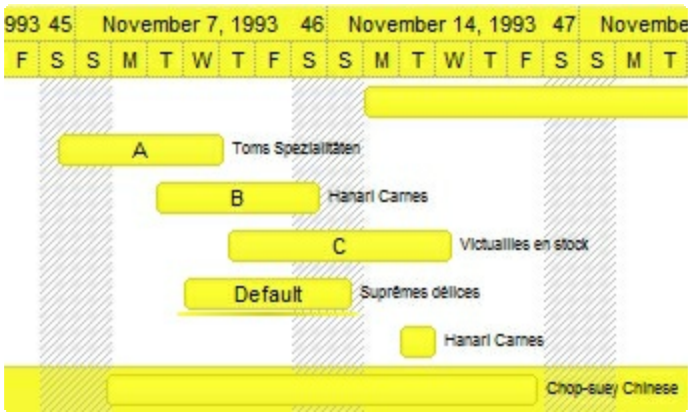
*The following picture shows the control with the RenderType property on 0x**40**00FFFF (25% Yellow, 0x40 or 64 in decimal is 25% from 256):*



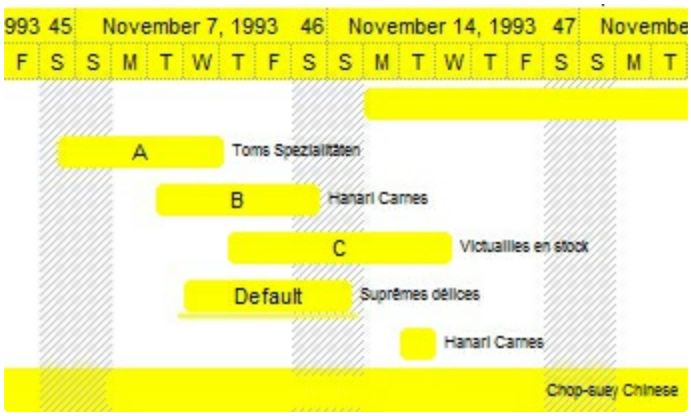
*The following picture shows the control with the RenderType property on 0x**80**00FFFF (50% Yellow, 0x80 or 128 in decimal is 50% from 256):*



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

Name	Description
Alignment	Specifies the column's alignment.
AllowDragging	Retrieves or sets a value indicating whether the user will be able to drag the column.
AllowGroupBy	Specifies if the column can be grouped by.
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.
DisplayFilterButton	Specifies whether the column's header displays the filter button.
	Specifies whether the drop down filter window displays a

DisplayFilterDate	date selector to specify the interval dates to filter for.
DisplayFilterPattern	Specifies whether the dropdown filterbar contains a textbox for editing the filter as pattern.
DisplaySortIcon	Retrieves or sets a value indicating whether the sort icon is visible on column's header, while the column is sorted.
Editor	Gets the column's editor object.
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 the filter type is exFilter, exPattern, exDate, exNumeric, exCheck or exImage
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 value of a cell hosted by column.
FormatColumn	Specifies the format to display the cells in the column.
FormatLevel	Retrieves or sets a value that indicates the layout of columns being displayed in the column's header.
GroupByFormatCell	Indicates the format of the cell to be displayed when the column gets grouped by.
GroupByTotalField	Indicates the aggregate formula to be displayed when the column gets grouped by.
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 in 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 a the column's key.
LevelKey	Retrieves or sets a value that indicates the key of the column's level.
MaxWidthAutoResize	Retrieves or sets a value that indicates the maximum column's width when the WidthAutoResize is True.
MinWidthAutoResize	Retrieves or sets a value that indicates the minimum column's width when the WidthAutoResize is True.
PartialCheck	Specifies whether the column supports partial check feature.
Position	Retrieves or sets a value that indicates the position of the column in the header bar area.
Selected	Retrieves or sets a value that indicates whether the cell in the column is selected.
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.
	Retrieves or sets a value indicating whether the column is

Visible	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 in the column's header.

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

Use the Alignment property to align cells in a column.By default the column is left aligned. 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. Use the HeaderImageAlignment property to align the image in the column's header, if it exists. Use the HeaderImage property to attach an icon to the column's header. Use the Def(exCellDrawPartsOrder) property to specify the order of the drawing parts inside the cell. The RightToLeft property automatically flips the order of the columns.

The following VB sample shows how you can display the cell's check box after the text:

```
With Grid1
   With .Columns.Add("Column")
      .Def(exCellHasCheckBox) = True
      .Def(exCellDrawPartsOrder) = "caption,check"
   End With
   With .Items
      .CellHasCheckBox(.AddItem("Caption 1"),0) = True
      .CellHasCheckBox(.AddItem("Caption 2"),0) = True
   End With
End With
```

The following VB.NET sample shows how you can display the cell's check box after the text:

```
With AxGrid1
   With .Columns.Add("Column")
      .Def(EXGRIDLib.DefColumnEnum.exCellHasCheckBox) = True
      .Def(EXGRIDLib.DefColumnEnum.exCellDrawPartsOrder) = "caption,check"
   End With
   With .Items
```

```
    .CellHasCheckBox(.AddItem("Caption 1"),0) = True
    .CellHasCheckBox(.AddItem("Caption 2"),0) = True
  End With
End With
```

The following C++ sample shows how you can display the cell's check box after the text:

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

  #import <ExGrid.dll>
  using namespace EXGRIDLib;
*/
EXGRIDLib::IGridPtr spGrid1 = GetDlgItem(IDC_GRID1)->GetControlUnknown();
EXGRIDLib::IColumnPtr var_Column = ((EXGRIDLib::IColumnPtr)(spGrid1->GetColumns()-
>Add(L"Column")));
  var_Column->PutDef(EXGRIDLib::exCellHasCheckBox,VARIANT_TRUE);
  var_Column->PutDef(EXGRIDLib::exCellDrawPartsOrder,"caption,check");
EXGRIDLib::IItemsPtr var_Items = spGrid1->GetItems();
  var_Items->PutCellHasCheckBox(var_Items->AddItem("Caption
1"),long(0),VARIANT_TRUE);
  var_Items->PutCellHasCheckBox(var_Items->AddItem("Caption
2"),long(0),VARIANT_TRUE);
```

The following C# sample shows how you can display the cell's check box after the text:

```
EXGRIDLib.Column var_Column = (axGrid1.Columns.Add("Column") as
EXGRIDLib.Column);
  var_Column.set_Def(EXGRIDLib.DefColumnEnum.exCellHasCheckBox,true);

var_Column.set_Def(EXGRIDLib.DefColumnEnum.exCellDrawPartsOrder,"caption,check");
EXGRIDLib.Items var_Items = axGrid1.Items;
  var_Items.set_CellHasCheckBox(var_Items.AddItem("Caption 1"),0,true);
  var_Items.set_CellHasCheckBox(var_Items.AddItem("Caption 2"),0,true);
```

The following VFP sample shows how you can display the cell's check box after the text:

```
with thisform.Grid1
```

```
  with .Columns.Add("Column")
    .Def(0) = .T.
    .Def(34) = "caption,check"
  endwith
  with .Items
    .DefaultItem = .AddItem("Caption 1")
    .CellHasCheckBox(0,0) = .T.
    .DefaultItem = .AddItem("Caption 2")
    .CellHasCheckBox(0,0) = .T.
  endwith
endwith
```

The following Delphi sample shows how you can display the cell's check box after the text:

```
with AxGrid1 do
begin
  with (Columns.Add('Column') as EXGRIDLib.Column) do
  begin
    Def[EXGRIDLib.DefColumnEnum.exCellHasCheckBox] := TObject(True);
    Def[EXGRIDLib.DefColumnEnum.exCellDrawPartsOrder] := 'caption,check';
  end;
  with Items do
  begin
    CellHasCheckBox[TObject(AddItem('Caption 1')),TObject(0)] := True;
    CellHasCheckBox[TObject(AddItem('Caption 2')),TObject(0)] := True;
  end;
end
```

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 AllowSizing property to allow user resizes a column at runtime.

property Column.AllowGroupBy as Boolean

Specifies if the column can be grouped by.

Type	Description
Boolean	A Boolean expression that specifies whether the user can drag and drop the column to be grouped by,

By default, the AllowGroupBy property is True. The AllowGroupBy property has effect only if the control's AllowGroupBy property is True. Use the AllowGroupBy property on False, to prevent a specific column to be sorted/grouped by. The same you can achieve if the AllowSort property is False. The SortBarVisible property specifies whether the control's sort bar is visible or hidden. If the control's sort bar is visible, the user can drag and drop columns to it, so the column get sorted and items grouped. The Group/Ungroup method groups or ungroup the control's list. For instance, you can remove the grouping items, by calling the Ungroup method. The GroupByTotalField property determines the formula to be applied to the column when it gets grouped. The GroupByFormatCell property determines the format of the cell to be displayed in the grouping item, when the column gets sorted.

property Column.AllowSizing as Boolean

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

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

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 AllowDragging property to forbid user to change the column's position by dragging. Use the Width property to specify the column's width. Use the ColumnsAllowSizing property to allow resizing the columns, when the control's header bar is not visible. Use the HeaderVisible property to show or hide the control's header bar.

property 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. The EnsureOnSort property prevents scrolling the control's content when the user sorts items.

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 value that indicates the required width of the column to fit the entire column's content.

Use the AutoWidth property to arrange the columns to fit the entire control's content. The AutoWidth property 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 EXGRIDLibCtl.Grid)
  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( CGrid& grid )
{
  grid.BeginUpdate();
  CColumns columns = grid.GetColumns();
  for ( long i = 0;i < columns.GetCount(); i++ )
  {
    CColumn column = columns.GetItem( COleVariant( i ) );
    if ( column.GetVisible() )
      column.SetWidth( column.GetAutoWidth() );
  }
  grid.EndUpdate();
```

```
}
```

The following VB.NET sample resizes all visible columns:

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

The following C# sample resizes all visible columns:

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

The following VFP sample resizes all visible columns:

```
with thisform.Grid1
   .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 in 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. Use the HeaderVertical property to display vertically the column's caption. Use the HeaderImage property to assign an icon to a column. 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. Use the FormatLevel property to display multiple levels in the column's header.

property Column.ComputedField as String

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

Type	Description
String	A String expression that indicates the formula to compute the field/cell. The formula is applied to all cells in the column with the CellValueFormat 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 CellValueFormat property on exText, uses the same formula to display their content. For instance, you can use the CellValueFormat property on exHTML, for cells in the column, that need to display other things than column's formula, or you can use the CellValueFormat property on exComputedField, to change the formula for a particular cell. Use the FormatColumn property to format the column.

Use the CellValueFormat property to change the type for a particular cell. Use the CellValue property to specify the cell's content. For instance, if the CellValueFormat property is exComputedField, the Caption property indicates the formula to compute the cell's content.

The Def(exCellValueFormat) 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(exCellValueFormat) 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.

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 indicates 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.

The expression supports cell's identifiers as follows:

- ***%0**, **%1**, **%2**, ... specifies the value of the cell in the column with the index 0, 1 2, ... The CellValue property specifies the cell's value. For instance, "%0 format ``" formats the value on the cell with the index 0, using current regional setting, while "int(%1)" converts the value of the column with the index 1, to integer.*
- ***%C0**, **%C1**, **%C2**, ... specifies the caption of the cell, or the string the cell displays in the column with the index 0, 1 2, ... The CellCaption property specifies the cell's caption. The cell's value may be different than what the cell displays as a string. For instance, let's say a cell display HTML format. The %0 returns the html format including the HTML tags, while %C0 returns the cell's content as string without HTML tags. For instance, "upper(%C1)" converts the caption of the cell with the index 1, to upper case, while "%C0 left 2" returns the leftmost two characters on the cell with the index 0.*
- ***%CD0**, **%CD1**, **%CD2**, ... 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 instance, "%CD0 = `your user data`" specifies all cells whose CellData property is `your user data`, on the column with the index 0.*
- ***%CS0**, **%CS1**, **%CS2**, ... specifies the cell's state in the column with the index 0, 1 2, ... The CellState property specifies the cell's state, and so it indicates whether the cell is checked or un-checked. For instance, "%CS0" defines all checked items on the column with the index 0, or "not %CS1" defines all un-checked items in the column with the index 1.*

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

property Column.CustomFilter as String

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

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

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

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

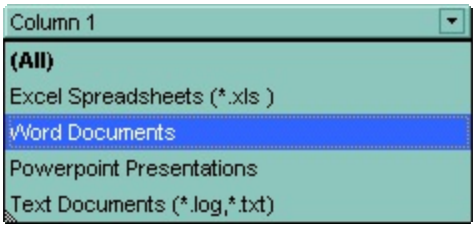
For example, you may have a list of documents and instead of listing the name of each document in the filter drop down list for the names column you may want to list the following:

Excel Spreadsheets
Word Documents
Powerpoint Presentations
Text Documents

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

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

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



property Column.Data as Variant

Associates an extra data to the column.

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

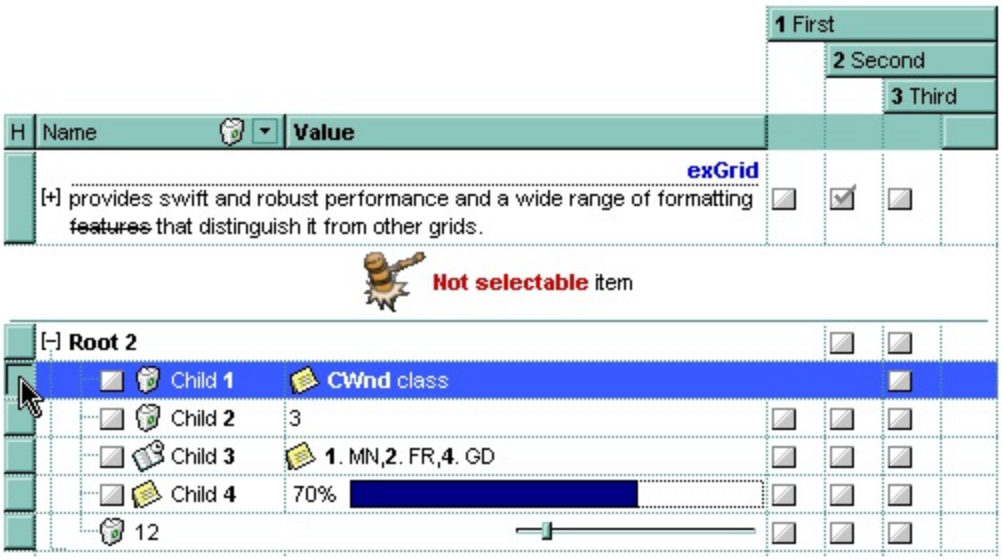
Use the Data property to assign any extra data to a column. Use the CellData property to assign an extra data to a cell. Use the ItemData property to assign an extra data to an item.

property Column.Def(Property as DefColumnEnum) as Variant

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

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

Use the Def property to specify a common value for given properties for all cells in the column. For instance, you can use the Def property to assign check boxes to all cells in the column, without enumerating them.



The following VB sample adds a header row column:

```
With Grid1.Columns.Add("H")
   .Def(exCellHasButton) = True
   .Position = 0
   .AllowDragging = False
   .HeaderAlignment = CenterAlignment
   .Width = 16
End With
```

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

```
Grid1.Columns(0).Def(exCellHasCheckBox) = True
```

The following C++ sample adds a header row column:

```
#include "Column.h"
#include "Columns.h"
CColumns columns = m_grid.GetColumns();
CColumn column( V_DISPATCH( &columns.Add("H") ) );
column.SetHeaderAlignment( 1 );
column.SetDef(2, COleVariant( VARIANT_TRUE ) );
column.SetPosition( 0 );
column.SetWidth( 16 );
column.SetAllowDragging( FALSE );
```

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

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

The following VB.NET sample adds a header row column:

```
With AxGrid1.Columns.Add("H")
   .Def(EXGRIDLib.DefColumnEnum.exCellHasButton) = True
   .Position = 0
   .AllowDragging = False
   .HeaderAlignment = EXGRIDLib.AlignmentEnum.CenterAlignment
   .Width = 16
End With
```

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

```
With AxGrid1.Columns(0)
   .Def(EXGRIDLib.DefColumnEnum.exCellHasCheckBox) = True
End With
```

The following C# sample adds a header row column:

```
EXGRIDLib.Columns columns = axGrid1.Columns;
EXGRIDLib.Column column = columns.Add("H") as EXGRIDLib.Column;
column.set_Def(EXGRIDLib.DefColumnEnum.exCellHasButton, true);
column.Position = 0;
column.HeaderAlignment = EXGRIDLib.AlignmentEnum.CenterAlignment;
```

```
column.AllowDragging = false;
column.Width = 16;
```

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

```
axGrid1.Columns[0].set_Def( EXGRIDLib.DefColumnEnum.exCellHasCheckBox, true );
```

The following VFP sample adds a header row column:

```
with thisform.Grid1.Columns.Add("H")
   .Position = 0
   .Def(2) = .t.
   .AllowDragging = .f.
   .Width = 16
endwith
```

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

```
with thisform.Grid1.Columns(0)
   .Def( 0 ) = .t.
endwith
```

property Column.DefaultSortOrder as Boolean

Specifies whether the default sort order is ascending or descending.

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

Use the DefaultSortOrder property to specify the default sort order, when the column's header is clicked. Use the SortOnClick property to specify when user can sort the columns by clicking the control's header. Use the SortOrder property to sort a column. Use the SortChildren method to sort items at runtime. 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 column's header displays a +/- (expanding button), to let user expands or collapse the column, when it is clicked.

By default, the DisplayExpandButton property is True. The DisplayExpandButton property displays the header's expanding button, only, if it contains child columns specified using the ExpandColumns property. The HasButtons property indicates the way the +/- (expanding button) is shown. Use the DisplayExpandButton property on True and ExpandColumns property to display the columns on multiple levels. The Expanded property expands programmatically a column. The control fires the LayoutChanged event when the user expands or collapse a column. Use the ExpandItem property to expand or collapse an item. The Index property indicates the column's index. The Visible property specifies whether a column is Visible or hidden.

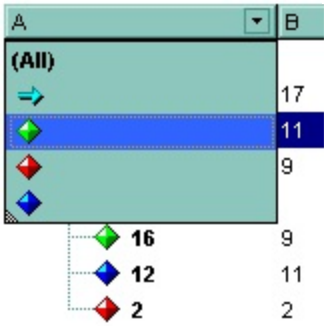
property Column.DisplayFilterButton as Boolean

Shows or hides the column's filter bar button.

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

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

The DisplayFilterPattern property determines whether the column's filter window includes the "Filter For" (pattern) field. Use the DisplayFilterDate property to include a date selector to the column's drop down filter window. Use the FilterBarDropDownHeight property to specify the height of the drop down filter window. Use the FilterBarDropDownWidth property to specify the width of the drop down filter window. Use the FilterType property to specify the type of the column's filter. Use the FilterType property to filter items based on the caption, check state or icons. Use the FilterList property to specify the list of items being included in the column's drop down filter list. Use the FilterInclude property to specify whether the child items should be included to the list when the user applies the filter. Use the Background(exHeaderFilterBarButton) property to change the visual appearance for the drop down filter button. Use the FilterCriteria property to filter items using the AND, OR and NOT operators. Use the CustomFilter property to define you custom filters. Use the ShowFilter method to show programmatically the column's drop down filter window.

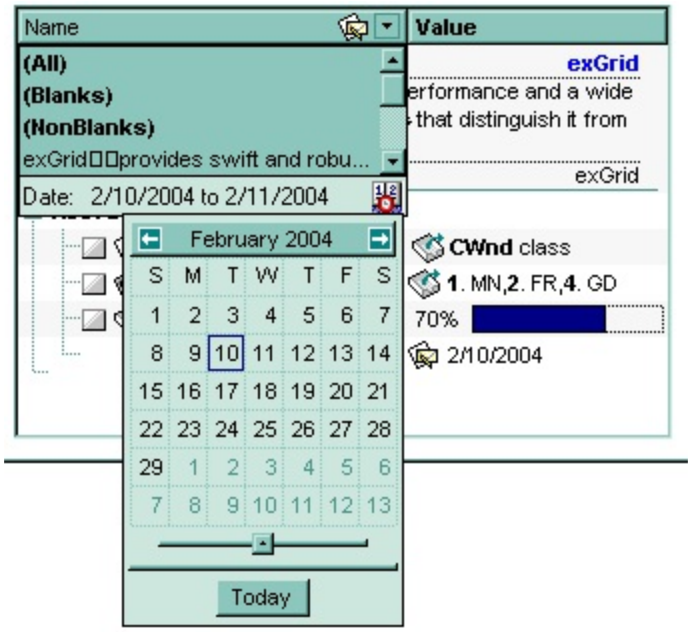


property Column.DisplayFilterDate as Boolean

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

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

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



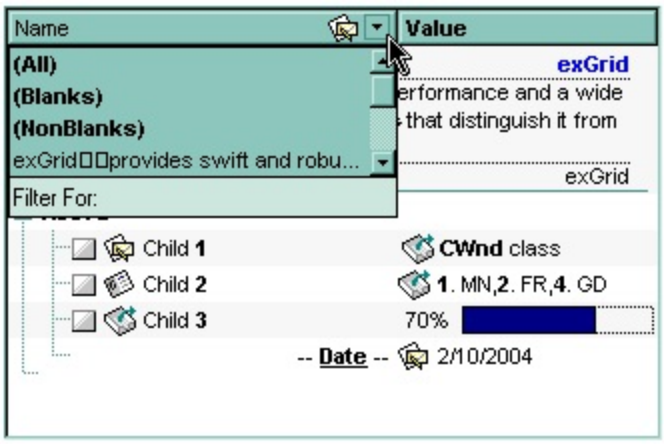
property Column.DisplayFilterPattern as Boolean

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

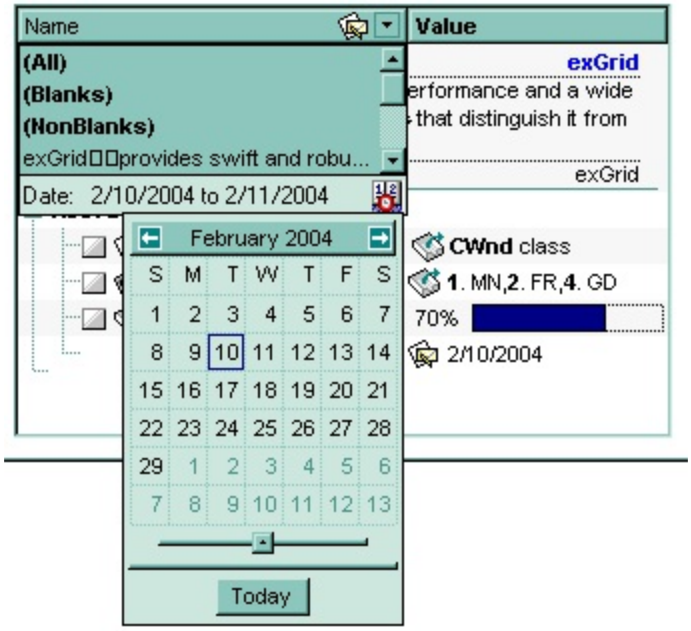
Type	Description
Boolean	A boolean expression that indicates whether the pattern field is visible or hidden.

Use the DisplayFilterButton property to show the column's filter button. If the DisplayFilterButton property is False the drop down filter window doesn't include the "Filter For" or "Date" field. Use the DisplayFilterDate property to filter items that match a given interval of dates. Use the FilterCriteria property to filter items using the AND, OR and 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.



The drop down filter window displays the "Date" field if the DisplayFilterPattern property is True, and the DisplayFilterDate property is True. If the drop down filter window displays "Date" field, and user types selects an interval of dates, the FilterType property of the Column is set to exDate, and Filter property of the Column object specifies the interval of dates being used in filtering.



Use the Description property to define the strings being displayed in the drop down filter window.

property Column.DisplaySortIcon as Boolean

Retrieves or sets a value indicating whether the sort icon is visible in 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, if the column was sorted by clicking in its header.

Use the DisplaySortIcon property to hide the icon of the column. Use the SortOnClick property to disable sorting columns by clicking in column's header. Use the SortChildren property of the Items object to sort by a column. Use the SortOrder property to sort a column. Use the SingleSort property to allow multiple sort columns.

property Column.Editor as Editor

Gets the column's editor object.

Type	Description
Editor	An Editor object that is associated to the column.

Use the Editor object to assign the same type of editor to all cells in the column. The Editor objects holds information about editing cells in the column. Use the EditType property to specify the column's edit type. Use the CellEditor property to assign a particular editor to a cell. Use the CellEditorVisible property to hide the cell's editor. Use the CellValue property to assign a value to a cell.

The following VB sample assigns a date editor to the first column:

```
With Grid1.Columns(0).Editor
   .EditType = DateType
End With
```

The following C++ sample assigns a date editor to the first column:

```
#include "Column.h"
#include "Columns.h"
CColumn column = m_grid.GetColumns().GetItem( COleVariant( long(0) ) );
CEditor editor = column.GetEditor();
editor.SetEditType( 7/*DateType*/ );
```

The following VB.NET sample assigns a date editor to the first column:

```
With AxGrid1.Columns(0).Editor
   .EditType = EXGRIDLib.EditTypeEnum.DateType
End With
```

The following C# sample assigns a date editor to the first column:

```
EXGRIDLib.Editor editor = axGrid1.Columns[0].Editor;
editor.EditType = EXGRIDLib.EditTypeEnum.DateType;
```

The following VFP sample assigns a date editor to the first column:

```
with thisform.Grid1.Columns.Item(0).Editor
```

```
    .EditType = 7 && DateType
endwith
```

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.

Use the Enabled property to disable a column. If a column is disabled, the user can select new items, but any checkbox, radio button, or editor in the cells of the column is disabled. Use the CellEnabled property to disable a particular cell. Use the EnableItem property to disable an item. Use the ReadOnly property to make your grid read only. Use the SelectableItem property to specify the user can select an item.

property Column.ExpandColumns as String

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

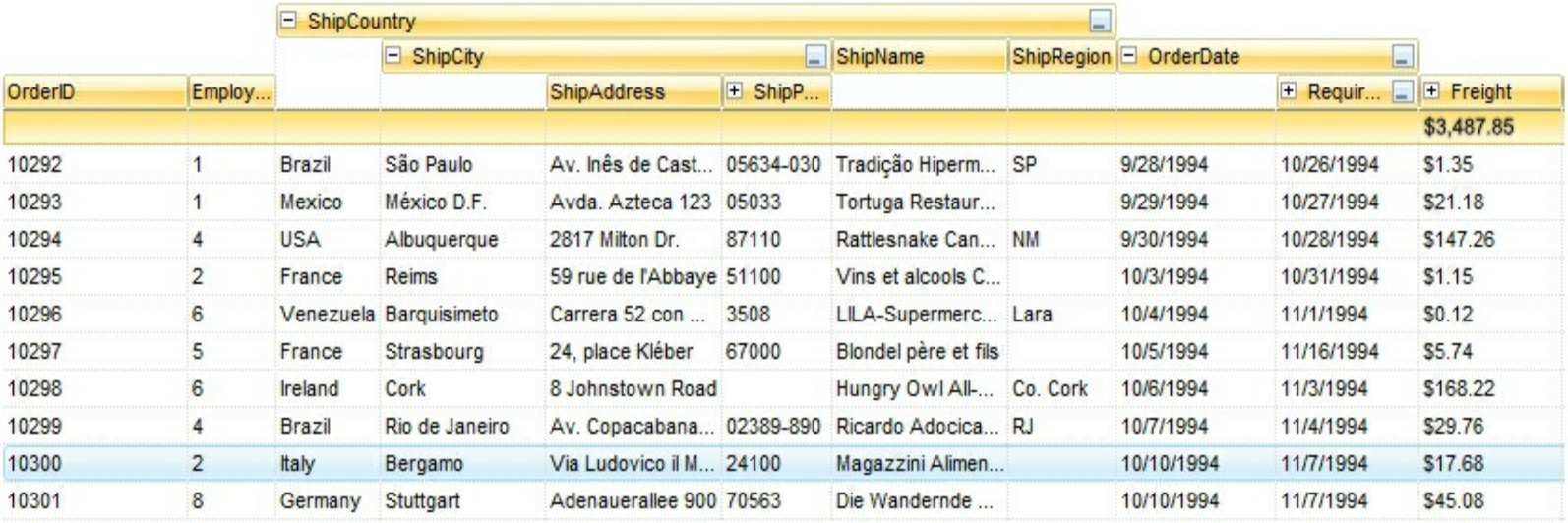
Type	Description
String	A String expression that specifies the list of columns to be shown/hidden when the current column is expanded or collapsed. The list indicates the index of each column to be shown/hidden separated by comma character. For instance, "2,3" indicates that the columns with the index 2 and 3 are displayed bellow the current column.

By default, the ExpandColumns property is empty. Use the ExpandColumns property to display the columns on multiple levels, or to allow your users to expand/collapse the columns. The DisplayExpandButton property specifies whether the column's header displays a +/- (expanding button), to let user expands or collapse the column, when it is clicked. The Expanded property expands programmatically a column. The control fires the LayoutChanged event when the user expands or collapse a column. Use the ExpandItem property to expand or collapse an item. The Index property indicates the column's index. The Visible property specifies whether a column is Visible or hidden.

The control performs showing/hiding the child columns as follow:

- If the column is expanded, the child columns are shown, and the current column is hidden, if the index of itself it is not included in the ExpandColumns property.
- If the column is collapsed, the recursively child columns are hidden, and the current column is shown.

The following screen shot shows the control's expandable header:



		ShipCountry								
			ShipCity			ShipName	ShipRegion	OrderDate		
OrderID	Employ...			ShipAddress	ShipP...				Requir...	Freight
										$3,487.85
10292	1	Brazil	São Paulo	Av. Inês de Cast...	05634-030	Tradição Hiperm...	SP	9/28/1994	10/26/1994	$1.35
10293	1	Mexico	México D.F.	Avda. Azteca 123	05033	Tortuga Restaur...		9/29/1994	10/27/1994	$21.18
10294	4	USA	Albuquerque	2817 Milton Dr.	87110	Rattlesnake Can...	NM	9/30/1994	10/28/1994	$147.26
10295	2	France	Reims	59 rue de l'Abbaye	51100	Vins et alcools C...		10/3/1994	10/31/1994	$1.15
10296	6	Venezuela	Barquisimeto	Carrera 52 con ...	3508	LILA-Supermerc...	Lara	10/4/1994	11/1/1994	$0.12
10297	5	France	Strasbourg	24, place Kléber	67000	Blondel père et fils		10/5/1994	11/16/1994	$5.74
10298	6	Ireland	Cork	8 Johnstown Road		Hungry Owl All-...	Co. Cork	10/6/1994	11/3/1994	$168.22
10299	4	Brazil	Rio de Janeiro	Av. Copacabana...	02389-890	Ricardo Adocica...	RJ	10/7/1994	11/4/1994	$29.76
10300	2	Italy	Bergamo	Via Ludovico il M...	24100	Magazzini Alimen...		10/10/1994	11/7/1994	$17.68
10301	8	Germany	Stuttgart	Adenauerallee 900	70563	Die Wandernde ...		10/10/1994	11/7/1994	$45.08

The following movie shows how you can use the Expandable Header support.

property Column.Expanded as Boolean

Expands or collapses the column.

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

The Expanded property expands programmatically a column. The ExpandColumns property specifies the list of columns to be shown when the current column is expanded. The DisplayExpandButton property specifies whether the column's header displays a +/- (expanding button), to let user expands or collapse the column, when it is clicked. The control fires the LayoutChanged event when the user expands or collapse a column. Use the ExpandItem property to expand or collapse an item. The Index property indicates the column's index. The Visible property specifies whether a column is Visible or hidden.

property Column.Filter as String

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

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

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

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

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

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

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

cell's checkbox. If the Filter property is empty, the filter is not applied to the column, when the ApplyFilter method is called. The drop down filter window displays the (All), (Checked) and (Unchecked) items.

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

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

The ApplyFilter method should be called to update the control's content after changing the Filter or FilterType property. The ClearFilter method clears the Filter and the FilterType properties. Use the FilterInclude property to specify whether the child items should be included to the list when the user applies the filter. Use the FilterCriteria property to filter items using the AND, OR and NOT operators. Use the CustomFilter property to define you custom filters. Use the ShowFilter method to show programmatically the column's drop down filter window.

property Column.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

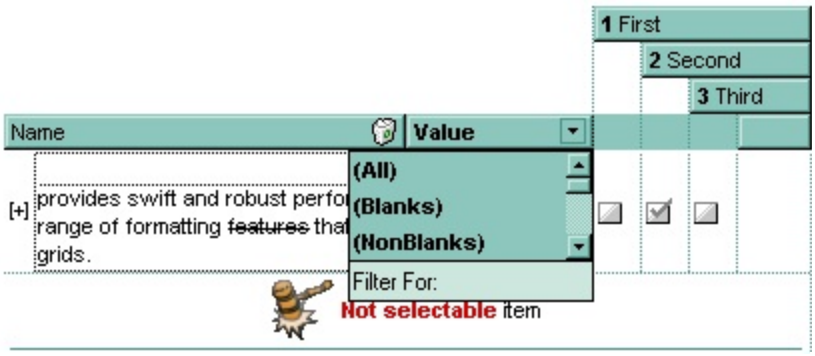
By default, the FilterBarDropDownWidth property is 1, and so, the width of the drop down filter window coincides with the width of the column. Use the Width property to specify the width of the column. Use FilterBarDropDownHeight property to specify the height of the drop down filter window. Use the 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.

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

```
With Grid1.Columns(0)
   .FilterBarDropDownWidth = 2
End With
```

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

```
With Grid1.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 FilterCriteria property to filter items using the AND, OR and NOT operators. Use the exSortItemsAsc flag to sort ascending the values in the drop down filter list. For instance, the **exAllItems OR exSortItemsAsc** specifies that the drop down filter window lists all items in ascending order. Add the exIncludeInnerCells flag if you require adding the inner cells value to the drop down filter window.

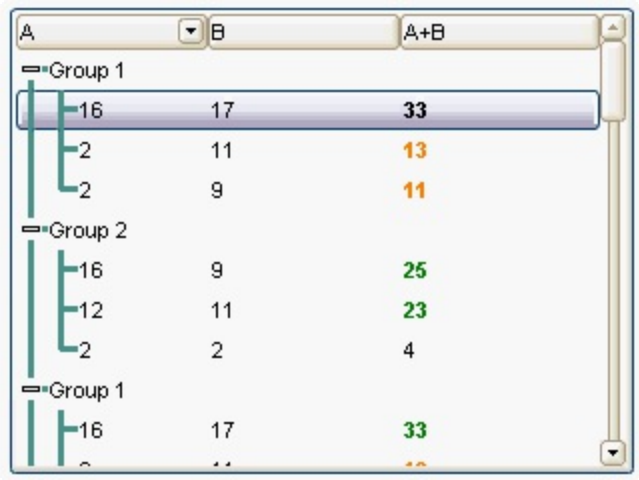
property Column.FilterOnType as Boolean

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

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

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

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



Steps:

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

property Column.FilterType as FilterTypeEnum

Specifies the column's filter type.

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

The FilterType property defines the filter's type. By default, the FilterType is exAll. No filter is applied if the FilterType is exAll. The Filter property defines the column's filter. Use the DisplayFilterButton property to display the column's filter button. Use the FilterInclude property to specify whether the child items should be included to the list when the user applies the filter. Use the FilterCriteria property to filter items using the AND, OR and NOT operators.

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.

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.

property Column.FireFormatColumn as Boolean

Retrieves or sets a value that indicates whether the control fires FormatColumn to format the value of a cell hosted by column.

Type	Description
Boolean	A boolean expression that indicates whether the control fires FormatColumn to format the value of a cell hosted by column

By default, the FireFormatColumn property is False. The FormatColumn event is fired only if the FireFormatColumn property of the Column object is True. The FormatColumn event lets the user to provide the cell's caption before it is displayed on the control's list. For instance, the FormatColumn event is useful when the column cells contains prices (numbers), and you want to display that column formatted as currency, like $50 instead 50. Also, it is useful to use the FormatColumn event when displaying computed cells.

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.

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 CellValueFormat 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 FormatCell property indicates the individually predefined format to be applied to particular cells. The FormatColumn and FormatCell properties support auto-numbering functions like explained bellow. The ComputedField property indicates the formula of the computed column.

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):

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

The **value** keyword in the FormatColumn property indicates the value being formatted.

The expression supports cell's identifiers as follows:

- ***%0**, **%1**, **%2**, ... specifies the value of the cell in the column with the index 0, 1 2, ... The CellValue property specifies the cell's value. For instance, "%0 format ``" formats the value on the cell with the index 0, using current regional setting, while "int(%1)" converts the value of the column with the index 1, to integer.*
- ***%C0**, **%C1**, **%C2**, ... specifies the caption of the cell, or the string the cell displays in the column with the index 0, 1 2, ... The CellCaption property specifies the cell's caption. The cell's value may be different than what the cell displays as a string. For instance, let's say a cell display HTML format. The %0 returns the html format including the HTML tags, while %C0 returns the cell's content as string without HTML tags. For instance, "upper(%C1)" converts the caption of the cell with the index 1, to upper case, while "%C0 left 2" returns the leftmost two characters on the cell with the index 0.*
- ***%CD0**, **%CD1**, **%CD2**, ... 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 instance, "%CD0 = `your user data`" specifies all cells whose CellData property is `your user data`, on the column with the index 0.*
- ***%CS0**, **%CS1**, **%CS2**, ... specifies the cell's state in the column with the index 0, 1 2, ... The CellState property specifies the cell's state, and so it indicates whether the cell is checked or un-checked. For instance, "%CS0" defines all checked items on the column with the index 0, or "not %CS1" defines all un-checked items in the column with the index 1.*

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 Grid1
```

```
    .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 AxGrid1
    .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 'EXG2ANTTLib' for the library: 'ExGrid 1.0 Control Library'

    #import "C:\\Windows\\System32\\ExGrid.dll"
    using namespace EXG2ANTTLib;
*/
EXG2ANTTLib::IGridPtr spGrid1 = GetDlgItem(IDC_G2ANTT1)->GetControlUnknown();
((EXG2ANTTLib::IColumnPtr)(spGrid1->GetColumns()->Add(L"Currency")))-
>PutFormatColumn(L"currency(dbl(value))");
EXG2ANTTLib::IItemsPtr var_Items = spGrid1->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:

```
(axGrid1.Columns.Add("Currency") as EXG2ANTTLib.Column).FormatColumn =
"currency(dbl(value))";
EXG2ANTTLib.Items var_Items = axGrid1.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:

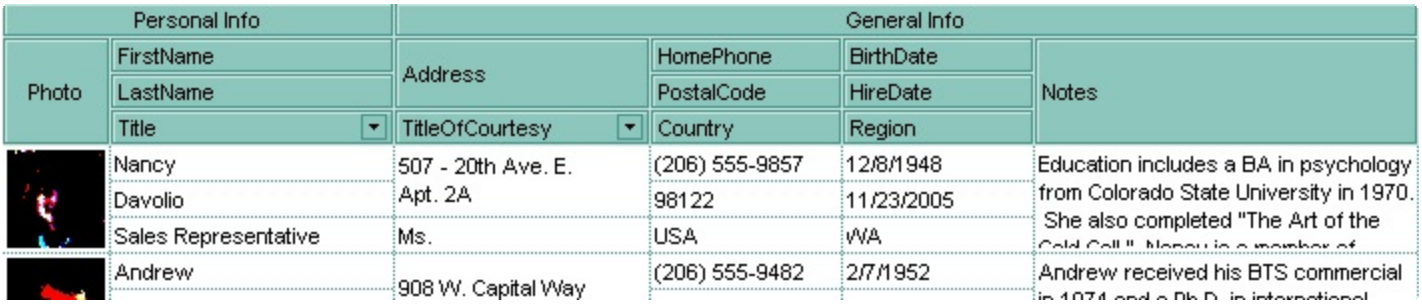
```
with thisform.Grid1
.Columns.Add("Currency").FormatColumn = "currency(dbl(value))"
with .Items
.AddItem("1.23")
.AddItem("2.34")
.AddItem("0")
.AddItem(5)
.AddItem("10000.99")
endwith
endwith
```

property Column.FormatLevel as String

Retrieves or sets a value that indicates the layout of columns being displayed in the column's header.

Type	Description
String	A string expression that indicates a CRD string that layouts the column's header. The Index elements in the CRD strings indicates the index of the column being displayed. The Caption elements in the CRD string support built-in HTML format.

By default, the FormatLevel property is empty. The FormatLevel property indicates the layout of the column in the control's header bar. Use the HeaderVisible property to show or hide the control's header bar. Use the HeaderHeight property to specify the height of the level in the control's header bar. Use the FormatLevel property to display multiple levels in the column's header. Use the LevelKey property to display neighbor columns on multiple levels. If the FormatLevel property is empty, the control displays the Caption or the HTMLCaption of the column. If the FormatLevel property is not empty it indicates the layout of the column being displayed. For instance, the FormatLevel = "1,2" indicates that the column's header is horizontally divided such as the left part displays the caption of the first column, and the right part displays the caption of the second column. Use the Visible property to specify whether a column is visible or hidden. Use the Add method to add new columns to the control. Use the DataSource property to bound the control to a recordset. Use the Def(exCellFormatLevel) property to specify the layout for all cells in the same column. Use the CellFormatLevel property to indicate the layout for a specific cell.



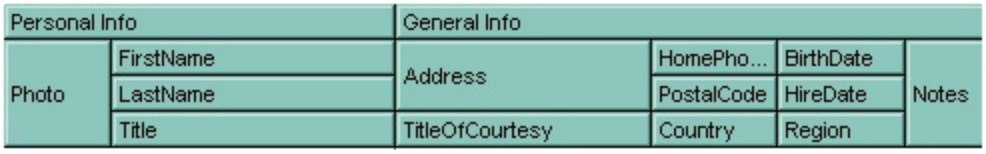
The following VB sample arranges the columns as in the above screen shot (the sample hides the columns and add instead two new columns { Personal Info, General Info }, where the layout is displayed.

```
With Grid1
   .BeginUpdate
   Dim c As EXGRIDLibCtl.Column
   For Each c In .Columns
      c.Visible = False
```

```
    Next
    With .Columns.Add("Personal Info")
        .AllowSort = False
        .AllowDragging = False
        .Width = 196
        .FormatLevel = "18;17/(14:54,(2/1/3))"
    End With
    With .Columns.Add("General Info")
        .AllowSort = False
        .AllowDragging = False
        .Width = 382
        .FormatLevel = "18;18/((7/18;4):128,((((12/10/11),(5/6/9)),15)))"
    End With
    .EndUpdate
End With
```

Before running the sample the control's header bar looks like follows:

After running the sample the control's header bar looks like follows:



The following C++ sample arranges the columns as in the above screen shot (the sample hides the columns and add instead two new columns { Personal Info, General Info }, where the layout is displayed.

```
m_grid.BeginUpdate();
CColumns cols = m_grid.GetColumns();
long nCount = cols.GetCount();
for ( long i = 0; i < nCount; i++ )
    cols.GetItem( COleVariant(i) ).SetVisible( FALSE );

CColumn col1( V_DISPATCH( &cols.Add( "Personal Info" ) ) );
col1.SetAllowSort( FALSE );
col1.SetAllowDragging( FALSE );
col1.SetWidth( 196 );
```

```
col1.SetFormatLevel( "18;18/(15:54,(2/1/4))" );
CColumn col2( V_DISPATCH( &cols.Add( "General Info" ) ) );
col2.SetAllowSort( FALSE );
col2.SetAllowDragging( FALSE );
col2.SetWidth( 512 );
col2.SetFormatLevel( "18;19/((8/18;5):128,((((13/11/12),(6/7/10)),16)))" );
m_grid.EndUpdate();
```

The following VB.NET sample arranges the columns as in the above screen shot (the sample hides the columns and add instead two new columns { Personal Info, General Info }, where the layout is displayed.

```
With AxGrid1
   .BeginUpdate()
   Dim c As EXGRIDLib.Column
   For Each c In .Columns
      c.Visible = False
   Next
   With .Columns.Add("Personal Info")
      .AllowSort = False
      .AllowDragging = False
      .Width = 196
      .FormatLevel = "18;18/(15:54,(2/1/4))"
      .Def(EXGRIDLib.DefColumnEnum.exCellFormatLevel) = "15:54,(2/1/4)"
   End With
   With .Columns.Add("General Info")
      .AllowSort = False
      .AllowDragging = False
      .Width = 512
      .FormatLevel = "18;19/((8/18;5):128,((((13/11/12),(6/7/10)),16)))"
      .Def(EXGRIDLib.DefColumnEnum.exCellFormatLevel) = "(8/18;5):128,((((13/11/12),
(6/7/10)),16))"
   End With
   .EndUpdate()
End With
```

The following C# sample arranges the columns as in the above screen shot (the sample hides the columns and add instead two new columns { Personal Info, General Info }, where

the layout is displayed.

```
axGrid1.BeginUpdate();
foreach( EXGRIDLib.Column c in axGrid1.Columns)
   c.Visible = false;
EXGRIDLib.Column c1 = axGrid1.Columns.Add("Personal Info") as EXGRIDLib.Column;
c1.AllowSort = false;
c1.AllowDragging = false;
c1.Width = 196;
c1.FormatLevel = "18;18/(15:54,(2/1/4))";
c1.set_Def(EXGRIDLib.DefColumnEnum.exCellFormatLevel,"15:54,(2/1/4)");

EXGRIDLib.Column c2 = axGrid1.Columns.Add("General Info") as EXGRIDLib.Column;
c2.AllowSort = false;
c2.AllowDragging = false;
c2.Width = 512;
c2.FormatLevel = "18;19/((8/18;5):128,((((13/11/12),(6/7/10)),16)))";
c2.set_Def(EXGRIDLib.DefColumnEnum.exCellFormatLevel,"(8/18;5):128,((((13/11/12),
(6/7/10)),16))");
axGrid1.EndUpdate();
```

The following VFP sample arranges the columns as in the above screen shot (the sample hides the columns and add instead two new columns { Personal Info, General Info }, where the layout is displayed.

```
with thisform.Grid1
   .BeginUpdate()
   with .Columns
      for i = 0 to .Count - 1
      .Item(i).Visible = .f.
   next
   with .Add("Personal Info")
      .AllowSort = .f.
      .AllowDragging = .f.
      .Width = 196
      .FormatLevel = "18;18/(15:54,(2/1/4))"
      .Def(32) = "15:54,(2/1/4)"
   endwith
```

```
  with .Add("General Info")
    .AllowSort = .f.
    .AllowDragging = .f.
    .Width = 512
    .FormatLevel = "18;19/((8/18;5):128,((((13/11/12),(6/7/10)),16)))"
    .Def(32) = "(8/18;5):128,((((13/11/12),(6/7/10)),16))"
  endwith
  endwith
  .EndUpdate()
endwith
```

property Column.GroupByFormatCell as String

Indicates the format of the cell to be displayed when the column gets grouped by.

Type	Description
String	A String expression that may specify HTML format, **<caption>** and **value** keywords as explained bellow.

By default, the GroupByFormatCell property is "'<caption> (' + value + ')'", which indicates that the grouping label is shown in bold, followed by the computed value of the GroupByTotalField property. The GroupByFormatCell property determines the format of the cell to be displayed in the grouping item, when the column gets sorted. The GroupByTotalField property determines the formula to be applied to the column when it gets grouped. *When the control is performing a group-by operation, the Items.FormatCell(Item,Items.GroupItem(Item)) property is set on GroupByFormatCell property, where the Item is the handle of the item being added during grouping or the Item parameter of the AddGroupItem event.*

In conclusion,

- *the **<caption>** keyword in the GroupByFormatCell property is replaced with the grouping label/value, and the result expression is passed to the FormatCell property.*
- *the **value** keyword indicates the computed value of the GroupByTotalField property.*

For instance:

- the *"'<caption> (' + currency(value) + ')'"* displays the grouping label, and the aggregate field as a currency, as specified in the regional settings.
- the *"'<caption> (' + currency(value) + `, inc. VAT ` + currency(1.19*value) + ')'"* displays the grouping label, and the aggregate field, including a computed field (VAT) as a currency, as specified in the regional settings.
- the *"'<caption> <fgcolor=808080>(Total ' + (value format ``) + ') </fgcolor>'"* displays the grouping label, and the aggregate field as a current number format, as specified in the regional settings, with a different font and foreground color.

The **value** keyword in the GroupByFormatCell property indicates the value to be formatted (as a result of the GroupByTotalField property):

The expression supports cell's identifiers as follows:

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

converts the value of the column with the index 1, to integer.

- ***%C0**, **%C1**, **%C2**, ... specifies the caption of the cell, or the string the cell displays in the column with the index 0, 1 2, ... The CellCaption property specifies the cell's caption. The cell's value may be different than what the cell displays as a string. For instance, let's say a cell display HTML format. The %0 returns the html format including the HTML tags, while %C0 returns the cell's content as string without HTML tags. For instance, "upper(%C1)" converts the caption of the cell with the index 1, to upper case, while "%C0 left 2" returns the leftmost two characters on the cell with the index 0.*
- ***%CD0**, **%CD1**, **%CD2**, ... 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 instance, "%CD0 = `your user data`" specifies all cells whose CellData property is `your user data`, on the column with the index 0.*
- ***%CS0**, **%CS1**, **%CS2**, ... specifies the cell's state in the column with the index 0, 1 2, ... The CellState property specifies the cell's state, and so it indicates whether the cell is checked or un-checked. For instance, "%CS0" defines all checked items on the column with the index 0, or "not %CS1" defines all un-checked items in the column with the index 1.*

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

property Column.GroupByTotalField as String

Indicates the aggregate formula to be displayed when the column gets grouped by.

Type	Description
String	A String expression that indicates the formula to be displayed on the grouping caption.

By default, the GroupByTotalField property is "count(current,rec,1)", which count recursively leaf items (implies recursively leaf items) of the grouping item. At runtime, the computed value of this formula is replaced in the HTML format being specified by the GroupByFormatCell property, for the **value** keyword. *When the control is performing a group-by operation, the Items.CellValue(Item,Items.GroupItem(Item)) property is set on GroupByTotalField property, and the Items.CellValueFormat(Item,Items.GroupItem(Item)) is exHTML + exTotalField (5), where the Item is the handle of the item being added during grouping or the Item parameter of the AddGroupItem event.* The GroupByTotalField property determines the formula to be applied to the column when it gets grouped. The GroupByFormatCell property determines the format of the cell to be displayed in the grouping item, when the column gets sorted.

For instance

- "*count(current,dir,1)*" counts the number of child items (not implies recursively child items).
- "*count(current,all,1)*" counts the number of all child items (implies recursively child items).
- "*sum(parent,dir,%1=0?0:1)*" counts the not-zero values in the second column (%1)
- "*sum(parent,dir,%1 + %2)*" indicates the sum of all cells in the second (%1) and third (%2) column that are directly descendent from the parent item.
- "*sum(all,rec,%1 + %2)*" sums all leaf cells in the second (%1) and third (%2) columns.

The syntax for the GroupByTotalField property property should be:
aggregate(**list**,**direction**,**formula**) where:

aggregate must be one of the following:

- *sum* - calculates the sum of values.
- *min* - retrieves the minimum value.
- *max* - retrieves the maximum value.
- *count* - counts the number of items.
- *avg* - calculates the average of values.

list must be one of the following:

- a *long* expression that specifies the index of the item being referred.
- a predefined string expression as follows:
  - *all* - indicates all items, so the formula is being applied to all items. The direction has no effect.
  - *current* - the current item.
  - *parent* - the parent item.
  - *root* - the root item.

direction must be one of the following:

- *dir* - collects the direct descendents.
- *rec* - collects the leaf descendents (leaf items). A leaf item is an item with no child items.
- *all* - collects all descendents.

Currently, the following items are excluded by aggregate functions:

- *not-sortable items*. The SortableItem property specifies whether the item can be sorted (a sortable item can change its position after sorting, while a not-sortable item keeps its position after sorting.
- *not-selectable items*. The SelectableItem property specifies whether the user can selects/focus the specified item.
- *divider items*. The ItemDivider property specifies whether the item displays a single cell, instead displaying whole cells.

In conclusion, aggregate functions counts ONLY items that are:

- *sortable*, SortableItem is True, by default.
- *selectable*, SelectableItem is True, by default.
- *not divider*, ItemDivider is -1, by default.

Shortly, by setting to a different value to any of these properties, makes the item to be ignored by the aggregate functions.

For instance

- count(current,dir,1) counts the number of child items (not implies recursively child items).
- count(current,all,1) counts the number of all child items (implies recursively child items).
- count(current,rec,1) counts the number of leaf items (implies recursively leaf items).
- count(current,rec,1) counts the number of leaf items (a leaf item is an item with no child items).
- sum(parent,dir,%1=0?0:1) counts the not-zero values in the second column (%1)

- sum(parent,dir,%1 + %2) indicates the sum of all cells in the second (%1) and third (%2) column that are directly descendent from the parent item.
- sum(all,rec,%1 + %2) sums all leaf cells in the second (%1) and third (%2) columns.

property Column.HeaderAlignment as AlignmentEnum

Specifies the alignment of the column's caption.

Type	Description
AlignmentEnum	An AlignmentEnum expression that indicates the alignment of the 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. The RightToLeft property automatically flips the order of the columns

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 icon in the Images collection, that's displayed on the column's header. The last 7 bits in the high significant byte of the long expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part.

Use the HeaderImage property to add an icon to the column's header. The HeaderImage property does not set the icon for any of the column cells. Use the CellImage property to set an icon for a particular cell. Use the HeaderImageAlignment property to align the icon in the column's header. If the index of the icon in the column's header doesn't exist in the Images collection, no icon is displayed. Use the DisplaySortIcon property to specify whether the control displays the sorting icon when the user sorts a column. Use the Images method to assign a list of icons to the control at runtime. Use the <img> built-in HTML tag to insert multiple custom size picture/icons to the same header.

The following VB sample hides the icon in the column's header:

```
Grid1.Columns("Editor").HeaderImage = -1
```

The following C++ sample hides the icon in the header of the first column:

```
#include "Column.h"
#include "Columns.h"
CColumn column = m_grid.GetColumns().GetItem( COleVariant( long(0) ) );
column.SetHeaderImage( -1 );
```

The following VB.NET sample hides the icon in the header of the first column:

```
With AxGrid1.Columns(0)
   .HeaderImage = -1
End With
```

The following C# sample hides the icon in the header of the first column:

```
EXGRIDLib.Column column = axGrid1.Columns[0];
column.HeaderImage = -1;
```

The following VFP sample hides the icon in the header of the first column:

```
with thisform.Grid1.Columns.Item(0)
   .HeaderImage = -1
endwith
```

property Column.HeaderImageAlignment as AlignmentEnum

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

Type	Description
AlignmentEnum	An AlignmentEnum expression that indicates the alignment of the icon 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. The RightToLeft property automatically flips the order of the columns

property Column.HeaderItalic as Boolean

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

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

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

property Column.HeaderStrikeOut as Boolean

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

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

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

property Column.HeaderUnderline as Boolean

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

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

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

property Column.HeaderVertical as Boolean

Specifies whether the column's header is vertically displayed.

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

Use the HeaderVertical property to display vertically the column's caption. Use the 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. 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 HeaderImage property to assign an icon to a column. The list of built-in HTML tags supported are here. Use the FormatLevel property to display multiple levels in the column's header.

property Column.Index as Long

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

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

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

property Column.Key as String

Retrieves or sets a the column's key.

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

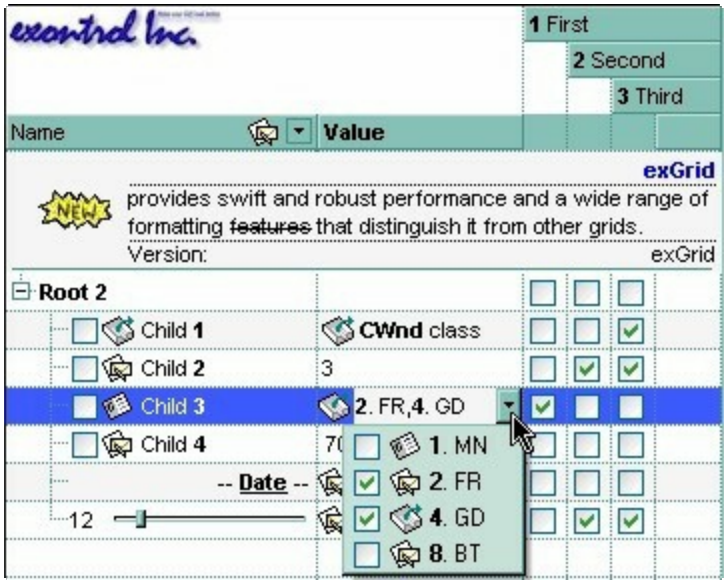
The column's key defines a column when using the Item property. Use the Index or the Key property to identify a column, when using the Columns property.

property Column.LevelKey as Variant

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

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

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



property Column.MaxWidthAutoResize as Long

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

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

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

property Column.MinWidthAutoResize as Long

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

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

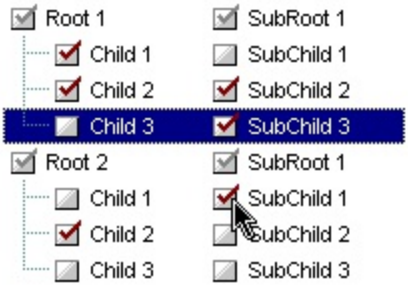
If the WidthAutoResize property is False, the MaxWidthAutoResize and MinWidthAutoResize properties have no effect. The MinWidthAutoResize property specifies the minimum column's width. The control recalculates the column's width each time when an item is expanded or collapsed. Use the WidthAutoResize,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

property Column.PartialCheck as Boolean

Specifies whether the column supports partial check feature.

Type	Description
Boolean	A boolean expression that indicates whether the column supports 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. The control fires the CellStateChanged event when the user clicks a checkbox or a radio button in 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 Visible property to hide a column. Use the Width property to specify the column's width.

The following VB6 sample enumerates the visible columns as they are displayed:

```
Private Sub enumColumns(ByVal g As EXGRIDLibCtl.Grid)
  Dim cArray() As EXGRIDLibCtl.Column
  With g
    ReDim Preserve cArray(.Columns.Count)
    For Each c In .Columns
      If (c.Visible) Then
        Set cArray(c.Position) = c
      End If
    Next
  End With
  For Each c In cArray
    If Not c Is Nothing Then
      Debug.Print c.Caption & "(" & c.Index & ")"
    End If
  Next
End Sub
```

property Column.Selected as Boolean

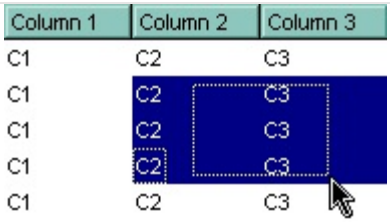
Retrieves or sets a value that indicates whether the cell in the column is selected.

Type	Description
Boolean	A boolean expression that specifies whether the cell in the column is selected.

Use the Selected property to determine the cells being selected, when FullRowSelect property is exRectSel. Use the SelectItem property to programmatically selects an item. Use the SingleSel property to allow multiple items or cells in the selection. The control fires the SelectionChanged event when user changes the selection.

The following VB sample copies the selected cells to the clipboard, if the FullRowSelect property is exRectSel:

```
Private Sub Grid1_SelectionChanged()
  Dim strData As String
  With Grid1
    Dim i As Long, h As HITEM
    For i = 0 To .Items.SelectCount - 1
      h = .Items.SelectedItem(i)
      Dim c As Column
      For Each c In .Columns
        If (c.Selected) Then
          strData = strData + .Items.CellCaption(h, c.Index) + vbTab
        End If
      Next
      strData = strData + vbCrLf
    Next
  End With
  Clipboard.Clear
  Clipboard.SetText strData
End Sub
```



The following C++ sample copies the selected cells to the clipboard, if the FullRowSelect property is exRectSel:

```
#include "Column.h"
#include "Columns.h"
#include "Items.h"
void OnSelectionChangedGrid1()
{
   CString strData;
   CColumns cols = m_grid.GetColumns();
   CItems items = m_grid.GetItems();
   for ( long i = 0; i < items.GetSelectCount(); i++ )
   {
      COleVariant vtItem( items.GetSelectedItem( i ) );
      for ( long j = 0; j < cols.GetCount(); j++ )
      {
         COleVariant vtColumn( j );
         if ( cols.GetItem( vtColumn ).GetSelected() )
            strData += items.GetCellCaption(vtItem, vtColumn ) + "\t";
      }
      strData += "\r\n";
   }
   if ( OpenClipboard() )
   {
      EmptyClipboard();
      HGLOBAL hGlobal = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE,
strData.GetLength() );
      CopyMemory( GlobalLock( hGlobal ), strData.operator LPCTSTR(),
strData.GetLength() );
      GlobalUnlock( hGlobal );
      SetClipboardData( CF_TEXT, hGlobal );
      CloseClipboard();
   }
}
```

The following VB.NET sample copies the selected cells to the clipboard, if the FullRowSelect property is exRectSel:

```
Private Sub AxGrid1_SelectionChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles AxGrid1.SelectionChanged
   Dim strData As String = ""
   With AxGrid1
      Dim i As Integer, h As Integer, j As Integer
      For i = 0 To .Items.SelectCount - 1
         h = .Items.SelectedItem(i)
         For j = 0 To .Columns.Count - 1
            Dim c As EXGRIDLib.Column = .Columns(j)
            If (c.Selected) Then
               strData = strData + .Items.CellCaption(h, c.Index) + vbTab
            End If
         Next
         strData = strData + vbCrLf
      Next
   End With
   Clipboard.Clear()
   Clipboard.SetText(strData)
End Sub
```

The following C# sample copies the selected cells to the clipboard, if the FullRowSelect property is exRectSel:

```
private void axGrid1_SelectionChanged(object sender, System.EventArgs e)
{
   string strData = "";
   for (int i = 0; i < axGrid1.Items.SelectCount; i++)
   {
      for ( int j = 0; j < axGrid1.Columns.Count; j++ )
         if (axGrid1.Columns[j].Selected)
         {
            string cellData =
axGrid1.Items.get_CellCaption(axGrid1.Items.get_SelectedItem(i), j);
            strData += cellData + "\t";
         }
      strData += "\r\n";
   }
```

```
    Clipboard.Clear();
    Clipboard.SetText(strData);
}
```

The following VFP sample copies the selected cells to the clipboard, if the FullRowSelect property is exRectSel (SelectionChanged event):

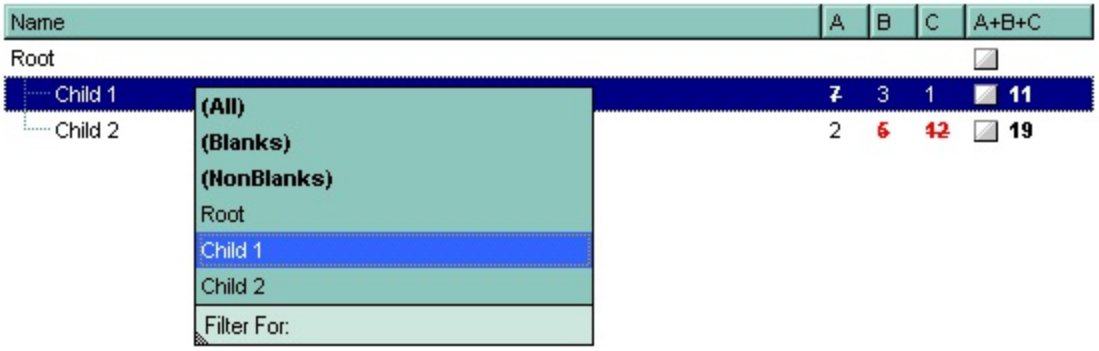
```
*** ActiveX Control Event ***

with thisform.Grid1.Items
    local strData, i, j, cols
    strData = ""
    cols = thisform.Grid1.Columns
    for i = 0 to .SelectCount - 1
        .DefaultItem = .SelectedItem( i )
        for j = 0 to cols.Count - 1
            if ( cols.Item(j).Selected )
                strData = strData + .CellCaption(0,j) + chr(9)
            endif
        next
        strData = strData + chr(13) + chr(10)
    next
    _CLIPTEXT = strData
endwith
```

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 Grid1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
   If (Button = 2) Then
      With Grid1.Columns
         With .Item(Grid1.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 AxGrid1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_MouseUpEvent) Handles AxGrid1.MouseUpEvent
   If (e.button = 2) Then
      With AxGrid1.Columns
         With .Item(AxGrid1.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 axGrid1_MouseUpEvent(object sender,
AxEXGRIDLib._IGridEvents_MouseUpEvent e)
{
   if (e.button == 2)
   {
      EXGRIDLib.Column c = axGrid1.Columns[axGrid1.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 OnMouseUpGrid1(short Button, short Shift, long X, long Y)
{
    m_grid.GetColumns().GetItem( COleVariant( m_grid.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.Grid1.Columns
        With .Item(thisform.Grid1.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 EnsureOnSort property prevents scrolling the control's content when the user sorts items. The SortableItem property specifies whether the item keeps its position after sorting.

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::

  ```
  Grid1.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

```
Grid1.Items.SortChildren 0, "Column 1", False
```

The SortType property of the Column object specifies the way how a column gets sorted. By default, a column gets sorted as string. If you need to sort your dates, the following snippet of code should be used:

```
With Grid1
    With .Columns(0)
      .SortType = SortDate
    End With
End With
```

If you need to sort a column using your special way you may want to use the SortType = SortUserData, or SortType = SortCellData that sorts the column using CellData / CellSortData property for each cell in the column. In this case, the CellData or CellSortData property holds numeric values only.

property Column.SortPosition as Long

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

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

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

property Column.SortType as SortTypeEnum

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

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

By default, the column's sort type is string. Use the SortType property to specify how the control sorts the column. Use the DisplaySortIcon property to hide the sort icon displayed when the column was sorted. Use the SortChildren method to sort items. Use the CellCaption property to get the string being displayed in the cell. Use the CellValue property to specify the cell's value. Use the CellSortData to specify the data being sorted when the SortType property is SortCellData or SortCellDataString. Use the CellData property to specify the values being sorted if the SortType property is SortUserData. 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 SingleSort property to specify whether the control supports sorting by single or multiple columns. The SortOrder property determines the column's sort order. The SortableItem property specifies whether the item keeps its position after sorting.

property Column.ToolTip as String

Specifes the column's tooltip description.

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

By default, the ToolTip property is "..." (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. The control fires the ToolTip event when the column's tooltip is about to be displayed. The ToolTipWidth property specifies a value that indicates the width of the tooltip window, in pixels.

property Column.Visible as Boolean

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

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

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

The following VB6 sample enumerates the visible columns as they are displayed:

```
Private Sub enumColumns(ByVal g As EXGRIDLibCtl.Grid)
  Dim cArray() As EXGRIDLibCtl.Column
  With g
    ReDim Preserve cArray(.Columns.Count)
    For Each c In .Columns
      If (c.Visible) Then
        Set cArray(c.Position) = c
      End If
    Next
  End With
  For Each c In cArray
    If Not c Is Nothing Then
      Debug.Print c.Caption & "(" & c.Index & ")"
    End If
  Next
End Sub
```

property Column.Width as Long

Retrieves or sets the column's width.

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

The Width property resizes a column at runtime. Use the AutoWidth property to compute the width that's required to fit all cells in the column. Use the WidthAutoResize property to automatically resize the column while the user expands or collapses items. Use the Visible property to hide a column. The ColumnAutoResize property specifies whether the visible columns fit the control's client area. If the ColumnAutoResize property is True, the Width property may not resize the column to the desired value, because all visible columns must fit the control's client area. By default, the control adds horizontal scroll bar when required. Use the ScrollBars property to add or remove the control's scroll bars. Use the Visible property to hide the column. 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 Grid1_AddColumn(ByVal Column As EXGRIDLibCtl.IColumn)
   Column.Width = 128
End Sub
```

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

```
Private Sub AxGrid1_AddColumn(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_AddColumnEvent) Handles AxGrid1.AddColumn
   e.column.Width = 128
End Sub
```

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

```
private void axGrid1_AddColumn(object sender,
AxEXGRIDLib._IGridEvents_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 OnAddColumnGrid1(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.

Use the WidthAutoResize property if you need to display the entire caption of each cell in the column. If the WidthAutoResize property is True, the user is not able to resize the column, so the AllowSizing property has no effect in this case. Use the ColumnAutoResize property to specify whether the control resizes the visible columns so they fit the control's client area. 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 HeaderVisible property to hide the control's header bar. Use the BeginUpdate and EndUpdate method to maintain performance while adding columns and items to the control. Use the MinWidthAutoResize property to specify the minimum column width, while the WidthAutoResize property is True.

The following VB sample adds a single column that's resized when the user expands or collapses an item:

```
Private Sub Form_Load()
  With Grid1
    .BeginUpdate
    .LinesAtRoot = True
    .Columns.Add("Column1").WidthAutoResize = True

    With .Items
      Dim h As HITEM
      h = .AddItem("Item 1")
      .InsertItem h, , "Item 2"
      h = .InsertItem(h, , "Item 3")
      .ExpandItem(.FindItem("Item 1")) = True
    End With
    .EndUpdate
  End With
```

```
End Sub
```

The following C++ sample adds a single column that's resized when the user expands or collapses an item:

```
#include "Items.h"
#include "Column.h"
#include "Columns.h"
COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
m_grid.BeginUpdate();
m_grid.SetLinesAtRoot( 1 );
CColumn column( V_DISPATCH( &m_grid.GetColumns().Add( "Column 1" ) ) );
column.SetWidthAutoResize( TRUE );
CItems items = m_grid.GetItems();
long hItem = items.AddItem( COleVariant( "Item 1" ) );
hItem = items.InsertItem( hItem, vtMissing, COleVariant( "Item 2" ) );
items.InsertItem( hItem, vtMissing, COleVariant( "Item 3" ) );
items.SetExpandItem( items.GetFindItem(COleVariant("Item 1"), vtMissing, vtMissing),
TRUE );
m_grid.EndUpdate();
```

The following VB.NET sample adds a single column that's resized when the user expands or collapses an item:

```
With AxGrid1
   .BeginUpdate()
   .LinesAtRoot = EXGRIDLib.LinesAtRootEnum.exLinesAtRoot
   .Columns.Add("Column1").WidthAutoResize = True

   With .Items
      Dim h As Integer = .AddItem("Item 1")
      .InsertItem(h, , "Item 2")
      h = .InsertItem(h, , "Item 3")
      .ExpandItem(.FindItem("Item 1")) = True
   End With
   .EndUpdate()
End With
```

The following C# sample adds a single column that's resized when the user expands or

collapses an item:

```
axGrid1.BeginUpdate();
axGrid1.LinesAtRoot = EXGRIDLib.LinesAtRootEnum.exLinesAtRoot;
EXGRIDLib.Column column = axGrid1.Columns.Add("Column 1") as EXGRIDLib.Column ;
column.WidthAutoResize = true;
EXGRIDLib.Items items = axGrid1.Items;
int hItem = items.AddItem("Item 1");
hItem = items.InsertItem(hItem, null, "Item 2");
items.InsertItem(hItem, null, "Item 2");
items.set_ExpandItem(items.get_FindItem("Item 1", null, null), true);
axGrid1.EndUpdate();
```

The following VFP sample adds a single column that's resized when the user expands or collapses an item:

```
with thisform.Grid1
   .BeginUpdate()
   .LinesAtRoot = .t.
   .Columns.Add("Column1").WidthAutoResize = .t.
   With .Items
      local h
      h = .AddItem("Item 1")
      .InsertItem(h, , "Item 2")
      h = .InsertItem(h, , "Item 3")
      .DefaultItem = .FindItem("Item 1")
      .ExpandItem(0) = .t.
   EndWith
   .EndUpdate()
endwith
```

Columns object

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

Name	Description
Add	Adds a Column object to the collection and returns a reference to the newly created object.
Clear	Removes all objects in a collection.
Count	Returns the number of objects in a collection.
Item	Returns a specific Column of the Columns collection.
ItemBySortPosition	Returns a Column object giving its sorting position.
Remove	Removes a specific member from the Columns collection.
SortBarColumn	Returns the Column from control's SortBar giving its position.
SortBarColumnsCount	Retrieves the count of Columns, in the control's SortBar

method Columns.Add (ColumnCaption as String)

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

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

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

By default, the control has no columns. Use Add method to add new columns to the control. If the control contains no columns, you cannot add new items to the control. Use the Remove method to remove a specific column. Each time when a column has been added to columns collection the control fires the AddColumn event. If the control's DataSource property points to an ADO recordset the user doesn't need to add columns to the control. Instead, the Add method can be used to add computed fields for instance. Use the AddItem, InsertItem, InsertControlItem, PutItems, DataSource properties to add new items to the control. Use the BeginUpdate and EndUpdate methods to prevent control from painting while adding columns or items. Use the Editor property to assign an editor to the cells in the column. Use the Def property to specify default setting for cells in the column. Use the FormatLevel property to display multiple levels in the column's header.

The following VB sample adds two new columns to the control:

```
With Grid1
   .Columns.Add "Column 1"
   With .Columns.Add("Column 2")
      With .Editor
         .EditType = CalculatorType
      End With
   End With
End With
```

The following C++ sample adds two new columns to the control:

```
#include "Column.h"
#include "Columns.h"
COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
CColumns columns = m_grid.GetColumns();
columns.Add( "Column 1" );
```

```
CColumn column( V_DISPATCH( &columns.Add( "Column 2" ) ) );
CEditor editor = column.GetEditor();
editor.SetEditType( 21 /*CalculatorType*/ );
```

The following VB.NET sample adds two new columns to the control:

```
With AxGrid1
   .Columns.Add("Column 1")
   Dim c As EXGRIDLib.Column = .Columns.Add("Column 2")
   With c.Editor
      .EditType = EXGRIDLib.EditTypeEnum.CalculatorType
   End With
End With
```

The following C# sample adds two new columns to the control:

```
EXGRIDLib.Column column = axGrid1.Columns.Add("Column 1") as EXGRIDLib.Column ;
column = axGrid1.Columns.Add("Column 2") as EXGRIDLib.Column;
column.Editor.EditType = EXGRIDLib.EditTypeEnum.CalculatorType;
```

The following VFP sample adds two new columns to the control:

```
with thisform.Grid1
   .Columns.Add("Column 1")
   With .Columns.Add("Column 2")
      with .Editor
         .EditType = 21 && CalculatorType
      endwith
   EndWith
endwith
```

method Columns.Clear ()

Removes all objects in a collection.

Type	Description

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

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

The following VB sample enumerates the columns in the control:

```
For Each c In Grid1.Columns
   Debug.Print c.Caption
Next
```

The following VB sample enumerates the columns in the control:

```
For i = 0 To Grid1.Columns.Count - 1
   Debug.Print Grid1.Columns(i).Caption
Next
```

The following VC sample enumerates the columns in the control:

```
#include "Columns.h"
#include "Column.h"
CColumns columns = m_grid.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 AxGrid1.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:

```
EXGRIDLib.Columns columns =axGrid1.Columns;
for ( int i = 0; i < columns.Count; i++ )
{
    EXGRIDLib.Column column = columns[i];
    System.Diagnostics.Debug.WriteLine( column.Caption );
}
```

The following VFP sample enumerates the columns in the control:

```
with thisform.Grid1.Columns
    for i = 0 to .Count - 1
        wait window nowait .Item(i).Caption
    next
endwith
```

property Columns.Item (Index as Variant) as Column

Returns a specific Column of the Columns collection.

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

Use the Item property to access to a specific column. The Count property counts the columns in the control. Use the Columns property to access the control's Columns collection. The SortBarColumn / SortBarColumnsCount properties can be used to enumerate the columns in the control's sort bar. The Visible property indicates whether the column is visible or hidden. The Position property specifies the position of the column. The user can change the column's position by drag and drop, so the position of the column can be changed at runtime. Instead the Index property is a read only property that gives the index of the column in the collection.

The following VB6 sample enumerates the visible columns as they are displayed:

```
Private Sub enumColumns(ByVal g As EXGRIDLibCtl.Grid)
  Dim cArray() As EXGRIDLibCtl.Column
  With g
    ReDim Preserve cArray(.Columns.Count)
    For Each c In .Columns
      If (c.Visible) Then
        Set cArray(c.Position) = c
      End If
    Next
  End With
  For Each c In cArray
    If Not c Is Nothing Then
      Debug.Print c.Caption & "(" & c.Index & ")"
    End If
  Next
End Sub
```

The Item property is the default property of the Columns object so the following statements are equivalents:

```
Grid1.Columns.Item ("Freight")
Grid1.Columns ("Freight")
```

The following VB sample enumerates the columns in the control:

```
For i = 0 To Grid1.Columns.Count - 1
   Debug.Print Grid1.Columns(i).Caption
Next
```

The following VC sample enumerates the columns in the control:

```
#include "Columns.h"
#include "Column.h"
CColumns columns = m_grid.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 AxGrid1.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:

```
EXGRIDLib.Columns columns =axGrid1.Columns;
for ( int i = 0; i < columns.Count; i++ )
{
   EXGRIDLib.Column column = columns[i];
   System.Diagnostics.Debug.WriteLine( column.Caption );
}
```

The following VFP sample enumerates the columns in the control:

```
with thisform.Grid1.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 SortBarColumn / SortBarColumnsCount properties can be used to enumerate the columns in the control's sort bar.

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

```
Dim s As String, i As Long, c As Column
i = 0
With Grid1.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_grid.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 AxGrid1
   Dim s As String, i As Integer, c As EXGRIDLib.Column
   i = 0
   With AxGrid1.Columns
      c = .ItemBySortPosition(i)
      While (Not c Is Nothing)
         s = s + """" & c.Caption & """ " & IIf(c.SortOrder =
EXGRIDLib.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;
EXGRIDLib.Column column = axGrid1.Columns.get_ItemBySortPosition( i );
while ( column != null )
{
   strOutput += column.Caption + " " + ( column.SortOrder ==
EXGRIDLib.SortOrderEnum.SortAscending ? "A" : "D" ) + " ";
   column = axGrid1.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.Grid1.Columns
   c = .ItemBySortPosition(i)
   do While (!isnull(c))
      with c
         s = s + "'" + .Caption
         s = s + "' " + Iif(.SortOrder = 1, "A", "D") + " "
         i = i + 1
      endwith
      c = .ItemBySortPosition(i)
   enddo
endwith
s = "Sort: " + s
wait window nowait s
```

method Columns.Remove (Index as Variant)

Removes a specific member from the Columns collection.

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

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

property Columns.SortBarColumn (Position as Variant) as Column

Returns the Column from control's SortBar giving its position.

Type	Description
Position as Variant	A long expression that specifies the position where the column is requested
Column	A Column object that specifies the sorted/grouped column at giving position, or empty if no column is found.

The SortBarColumn / SortBarColumnsCount properties can be used to enumerate the columns in the control's sort bar. Use the SortOrder property of the Column object on SortAscending / SortDescending to add a column to the sort bar, on SortNone to remove the column from the control's sort bar. 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. For instance, the SortBarColumnsCount counts the number of grouped columns, if the control's AllowGroupBy property is True.

property Columns.SortBarColumnsCount as Long

Retrieves the count of Columns, in the control's SortBar

Type	Description
Long	A long expression that specifies the number of columns being shown in the control's sort bar.

By default, the SortBarColumnsCount property is 0. The SortBarColumnsCount property counts the columns being shown in the sort bar. The SortBarColumn / SortBarColumnsCount properties can be used to enumerate the columns in the control's sort bar. Use the SortOrder property of the Column object on SortAscending / SortDescending to add a column to the sort bar, on SortNone to remove the column from the control's sort bar. 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. For instance, the SortBarColumnsCount counts the number of grouped columns, if the control's AllowGroupBy property is True.

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 Grid1.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_grid.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 AxGrid1.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):

```
EXGRIDLib.ConditionalFormat cf = axGrid1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Bold = true;
cf.ApplyTo = (EXGRIDLib.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.Grid1.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 Grid1.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_grid.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 AxGrid1.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):

```
EXGRIDLib.ConditionalFormat cf = axGrid1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Bold = true;
cf.ApplyTo = (EXGRIDLib.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.Grid1.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**, ... specifies the value of the cell in the column with the index 0, 1 2, ... The CellValue property specifies the cell's value. For instance, "%0 format ``" formats the value on the cell with the index 0, using current regional setting, while "int(%1)" converts the value of the column with the index 1, to integer.*
- ***%C0**, **%C1**, **%C2**, ... specifies the caption of the cell, or the string the cell displays in the column with the index 0, 1 2, ... The CellCaption property specifies the cell's caption. The cell's value may be different than what the cell displays as a string. For instance, let's say a cell display HTML format. The %0 returns the html format including the HTML tags, while %C0 returns the cell's content as string without HTML tags. For instance, "upper(%C1)" converts the caption of the cell with the index 1, to upper case, while "%C0 left 2" returns the leftmost two characters on the cell with the index 0.*

- ***%CD0**, **%CD1**, **%CD2**, ... 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 instance, "%CD0 = `your user data`" specifies all cells whose CellData property is `your user data`, on the column with the index 0.*
- ***%CS0**, **%CS1**, **%CS2**, ... specifies the cell's state in the column with the index 0, 1 2, ... The CellState property specifies the cell's state, and so it indicates whether the cell is checked or un-checked. For instance, "%CS0" defines all checked items on the column with the index 0, or "not %CS1" defines all un-checked items in the column with the index 1.*

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

Usage examples:

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:

```
Grid1.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_grid.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:

```
AxGrid1.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:

```
axGrid1.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.Grid1.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 Grid1.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 Grid1.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_grid.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 AxGrid1.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):

```
EXGRIDLib.ConditionalFormat cf = axGrid1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Italic = true;
cf.ApplyTo = (EXGRIDLib.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.Grid1.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 Grid1.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_grid.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 AxGrid1.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):

```
EXGRIDLib.ConditionalFormat cf = axGrid1.ConditionalFormats.Add("%1+%2<%0",null);
```

```
cf.Bold = true;
cf.ApplyTo = (EXGRIDLib.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.Grid1.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 Grid1.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_grid.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 AxGrid1.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):

```
EXGRIDLib.ConditionalFormat cf = axGrid1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Underline = true;
cf.ApplyTo = (EXGRIDLib.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.Grid1.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:

```
Grid1.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 Grid1.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_grid.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_grid.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:

```
AxGrid1.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 AxGrid1.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:

```
axGrid1.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):

```
EXGRIDLib.ConditionalFormat cf = axGrid1.ConditionalFormats.Add("%1+%2<%0",null);
cf.Bold = true;
cf.ApplyTo = (EXGRIDLib.FormatApplyToEnum)1;
```

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

```
thisform.Grid1.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.Grid1.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 Grid1.ConditionalFormats
   Debug.Print c.Expression
Next
```

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

```
Dim i As Integer
With Grid1.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_grid.GetConditionalFormats().GetCount(); i++ )
{
   CConditionalFormat cf = m_grid.GetConditionalFormats().GetItem( COleVariant( i ) );
   OutputDebugString( cf.GetExpression() );
}
```

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

```
Dim c As EXGRIDLib.ConditionalFormat
For Each c In AxGrid1.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 AxGrid1.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 (EXGRIDLib.ConditionalFormat c in axGrid1.ConditionalFormats)
   System.Diagnostics.Debug.Write(c.Expression);
```

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

```
for (int i = 0; i < axGrid1.ConditionalFormats.Count; i++)
   System.Diagnostics.Debug.Write(axGrid1.ConditionalFormats[i].Expression);
```

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

```
with thisform.Grid1.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 Grid1.ConditionalFormats
   Debug.Print c.Expression
Next
```

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

```
Dim i As Integer
With Grid1.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_grid.GetConditionalFormats().GetCount(); i++ )
{
   CConditionalFormat cf = m_grid.GetConditionalFormats().GetItem( COleVariant( i ) );
   OutputDebugString( cf.GetExpression() );
}
```

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

```
Dim c As EXGRIDLib.ConditionalFormat
```

```
For Each c In AxGrid1.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 AxGrid1.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 (EXGRIDLib.ConditionalFormat c in axGrid1.ConditionalFormats)
   System.Diagnostics.Debug.Write(c.Expression);
```

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

```
for (int i = 0; i < axGrid1.ConditionalFormats.Count; i++)
   System.Diagnostics.Debug.Write(axGrid1.ConditionalFormats[i].Expression);
```

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

```
with thisform.Grid1.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.

Editor object

The Editor object holds information about an editor. A cell or a column may have assigned an editor. Use the Editor property to access the column's editor. Use the CellEditor property to access the cell's editor. The Editor object supports the following properties and methods:

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

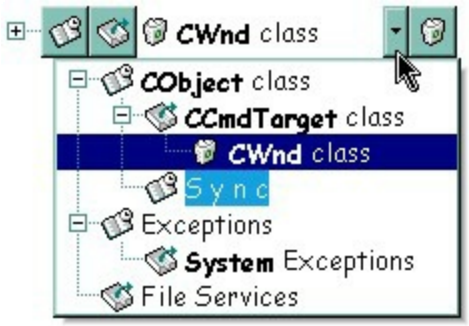
Mask	the editor.
MaskChar	Retrieves or sets a value that indicates the character used for masking.
Numeric	Specifies whether the editor enables numeric values only.
Option	Specifies an option for the editor.
PartialCheck	Retrieves or sets a value that indicates whether the associated check box has two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

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

Adds a new button to the editor with the given Key and aligned to the left or to the right side. You can specify the button's tooltip too.

Type	Description
Key as Variant	A Variant value that indicates the button's key. The ButtonClick event passes this value to Key parameter
Image as Variant	A long expression that indicates the index of button's icon. The index is valid for Images collection. By default the button has no icon associated.
Align as Variant	An AlignmentEnum expression that defines the button's alignment.
ToolTip as Variant	A string expression that indicates the the button's tooltip description. The tooltip shows up when cursor hovers the button. The ToolTip parameter may include buitl-in HTML tags.
ToolTipTitle as Variant	A string expression that indicates the tooltip's title.
ShortcutKey as Variant	A short expression that indicates the shortcut key being used to simulate clicking the button. The lower byte indicates the code of the virtual key, and the higher byte indicates the states for SHIFT, CTRL and ALT keys (last insignificant bits in the higher byte). The ShortcutKey expression could be 256 *((shift ? 1 : 0) + (ctrl ? 2 : 0) + (alt ? 4 : 0)) + vbKeyCode, For instance, a combination like CTRL + F3 is 256 * 2 + vbKeyF3, SHIFT + CTRL + F2 is 256 *(1 + 2) + vbKeyF2, and SHIFT + CTRL + ALT + F5 is 256 * (1 + 2 + 4) + vbKeyF5.

Use the AddButton method to add multiple buttons to the editor. Make sure that you are using unique keys for the buttons in the same editor, else the previous button is replaced. The editor doesn't allow two buttons with the same key. Use the ButtonWidth property to set the button's width. If the user clicks on one of the editor buttons, the ButtonClick event is fired. Use CellHasButton property to display's the cell's caption as a button. Use the RemoveButton method to remove a button that was previously added using the AddButton method. Use the ClearButtons method to clear the entire collection of buttons added with AddButton method. The control fires the ButtonClick event when the user clicks a button.



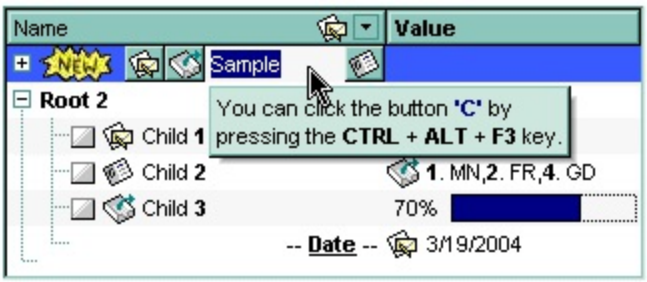
The following VB sample adds multiple buttons to the column's editor:

```
With Grid1
   With .Columns.Add("Column 1")
      .HeaderBold = True
      .HeaderImage = 1
      With .Editor
         .EditType = DropDownListType
         .DropDownAutoWidth = False

         .AddItem 0, "CS is bad", 3
         .AddItem 1, "xTras is the worst", 3
         .AddItem 2, "Yes, I agree!", 3

         .ButtonWidth = 24
         .AddButton "Key1", 1, , "This is a bit of text that should appear while the cursor is
over the button", "Information"
         .AddButton "Key2", 2
         .AddButton "Key3", 3, AlignmentEnum.RightAlignment
         .AddButton "Key3", 4, AlignmentEnum.RightAlignment
      End With
   End With
End With
```

The following VB sample adds an editor to the first visible item with three buttons, each of the button has a shortcut key to activate it using the keyboard:

```
With Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .ButtonWidth = 20
      .EditType = EditType
```

```
    .AddButton "A", 1, , "You can click the button <fgcolor=000080><b>'A'</b>
</fgcolor> by pressing the <b>F3</b> key.", , vbKeyF3
    .AddButton "B", 2, RightAlignment, "You can click the button <fgcolor=000080>
<b>'B'</b></fgcolor> by pressing the <b>CTRL + F3</b> key.", , vbKeyF3 + (256 * (2))
    .AddButton "C", 3, , "You can click the button <fgcolor=000080><b>'C'</b>
</fgcolor> by pressing the <b>CTRL + ALT + F3</b> key.", , vbKeyF3 + (256 * (2 + 4))
  End With
End With
```



The following C++ sample adds an EditType editor with a button to the first visible item:

```
#include "Items.h"
#include "Editor.h"
COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
CItems items = m_grid.GetItems();
CEditor editor = items.GetCellEditor( COleVariant( items.GetFirstVisibleItem() ),
COleVariant( long(0) ) );
editor.SetEditType( 1 /*EditType*/ );
editor.AddButton( COleVariant("A"), COleVariant("1"), vtMissing, vtMissing, vtMissing,
vtMissing  );
```

The following VB.NET sample adds an EditType editor with a button to the first visible item:

```
With AxGrid1.Items
  With .CellEditor(.FirstVisibleItem, 0)
    .EditType = EXGRIDLib.EditTypeEnum.EditType
    .AddButton("A", 1)
  End With
End With
```

The following C# sample adds an EditType editor with a button to the first visible item:

```
EXGRIDLib.Items items = axGrid1.Items;
```

```
EXGRIDLib.Editor editor = items.get_CellEditor(items.FirstVisibleItem, 0);
editor.EditType = EXGRIDLib.EditTypeEnum.EditType;
editor.AddButton("A", 1, null, null, null, null);
```

The following VFP sample adds an EditType editor with a button to the first visible item:

```
with thisform.Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = 1 && EditType
      .AddButton("A", 1)
   EndWith
endwith
```

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

Adds a new item to editor's predefined list.

Type	Description
Value as Long	A long expression that defines an unique predefined value
Caption as String	A string expression that specifies the HTML caption associated with the value. The format of the Caption parameter is "key|captionŚcaptionŚ...Ścaption", which indicates an item with the giving key / identifier, which displays multiple captions. The Caption allows using the following special characters: • | character (pipe, vertical bar, ALT + 126) defines the key or identifier of the item to add. Currently, the key is used by a **DropDownListType** editor to specify string codes rather numeric values for the cell's value (CellValue property) • Ś character (vertical broken bar, ALT + 221) defines captions for multiple columns. The Ś character can be escaped, so \Ś displays the Ś character (available for DropDownType, DropDownListType and PickEditType editors, 20.0+) For instance: • "New York City" defines the "**New York** City" item • "NYC|New York City" the "**New York** City" item with the "NYC" as key or identifier • "NYC|New York CityŚ783.8 km¸Ś8.42 mil" defines the "**New York** City" item with the "NYC" as key or identifier and sub-captions 783.8 km¸ and 8.42 mil (in separated columns) • "New York CityŚ783.8 km¸Ś8.42 mil" defines the "**New York** City" item and sub-captions 783.8 km¸ and 8.42 mil (in separated columns)
Image as Variant	A long expression that indicates the index of the item's icon (1-based). Use the Images method to assign a list of icons to the control.

Use the AddItem method to add new items to the editor's predefined list. Use the InsertItem method to insert child items to the editor's predefined list. If the AddItem method uses a Value already defined, the old item is replaced. The AddItem method has effect for the following type of editors: **DropDownType, DropDownListType, PickEditType,** and **CheckListType.** Check each EditType value for what Value argument should contain. Use the RemoveItem method to remove a particular item from the predefined list. Use the ClearItems method to clear the entire list of predefined values. Use the SortItems to sort the items. Use the ItemToolTip property to assign a tooltip to a predefined item into a drop down list. Use the Refresh method update immediately the cell's content when adding new items to a drop down list editor. The Caption parameter supports HTML tags listed here here.

The following VB sample adds items to a CheckListType editor:

```
With Grid1
   With .Columns.Add("CheckList").Editor
      .EditType = CheckListType
      .AddItem &H1, "ReadOnly", 1
      .AddItem &H2, "Hidden", 2
      .AddItem &H4, "System", 3
      .AddItem &H10, "Directory", 4
      .AddItem &H20, "Archive", 5
      .AddItem &H80, "Normal", 7
      .AddItem &H100, "Temporary", 8
   End With
.Items.AddItem &H1 + &H2
End With
```

The following VB sample adds predefined values to drop down list editor:

```
With Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = DropDownListType
      .AddItem 0, "No border", 1
      .AddItem 1, "Single Border", 2
      .AddItem 2, "Double Border", 3
   End With
End With
```

The following C++ sample adds predefined values to drop down list editor:

```
With Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = DropDownListType
      .AddItem 0, "No border", 1
      .AddItem 1, "Single Border", 2
      .AddItem 2, "Double Border", 3
   End With
End With
```

The following VB.NET sample adds predefined values to drop down list editor:

```
With AxGrid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = EXGRIDLib.EditTypeEnum.DropDownListType
      .AddItem(0, "No border", 1)
      .AddItem(1, "Single Border", 2)
      .AddItem(2, "Double Border", 3)
   End With
End With
```

The following C# sample adds predefined values to drop down list editor:

```
EXGRIDLib.Items items = axGrid1.Items;
EXGRIDLib.Editor editor = items.get_CellEditor(items.FirstVisibleItem, 0);
editor.EditType = EXGRIDLib.EditTypeEnum.DropDownListType ;
editor.AddItem(0, "No border", 1);
editor.AddItem(1, "Single border", 2);
editor.AddItem(2, "Double border", 3);
```

The following VFP sample adds predefined values to drop down list editor:

```
with thisform.Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = 3 && DropDownList
      .AddItem(0, "No border", 1)
      .AddItem(1, "Single Border", 2)
      .AddItem(2, "Double Border", 3)
```

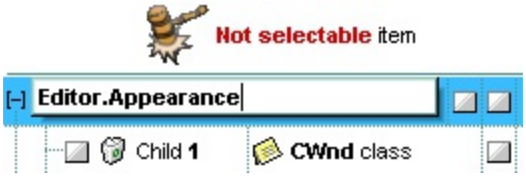
```
    EndWith
endwith
```

property Editor.Appearance as InplaceAppearanceEnum

Retrieves or sets the control's appearance

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

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



property Editor.ButtonWidth as Long

Specifies the width of the buttons in the editor.

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

Use the ButtonWidth property to increase or decrease the width of buttons in the editor. The button's height is the same with the ItemHeight property. If the ButtonWidth property is zero (0), the control hides the buttons. Use the DropDownVisible property to hide the editor's drop down button.

method Editor.ClearButtons ()

Clears the buttons collection.

Type	Description

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

method Editor.ClearItems ()

Clears the items collection.

Type	Description

The ClearItems method clears the predefined values that were added using the AddItem or InsertItem method. Use the RemoveItem method to remove a predefined value. Use the DropDownVisible property to hide the drop-down window.

method Editor.DropDown ()

Displays the drop down list.

Type	Description

The DropDown method shows the drop down portion of the cell's editor. The DropDown method has effect only if the editor has a drop down portion. The following editors have a drop down portion: DropDownType, DropDownListType, CheckListType, DateType, ColorType, FontType, PictureType, PickEditType, ColorListType, MemoDropDownType or CalculatorType. Use the AddItem, InsertItem method to add predefined value. Use the RemoveItem method to remove a predefined value. Use the DropDownVisible property to hide the drop down button, if it exists.

The following VB sample shows the drop down portion of an editor as soon as a cell is focused:

```
Private Sub Grid1_FocusChanged()
  With Grid1
    Dim i As Long
    i = .FocusColumnIndex
    With Grid1.Items
      If (.CellEditorVisible(.FocusItem, i)) Then
        Dim e As EXGRIDLibCtl.Editor
        Set e = Grid1.Columns(i).Editor
        If .HasCellEditor(.FocusItem, i) Then
          Set e = .CellEditor(.FocusItem, i)
        End If
        If Not e Is Nothing Then
          e.DropDown
        End If
      End If
    End With
  End With
End Sub
```

The following C++ sample shows the drop down portion of an editor as soon as a cell is focused:

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

```
#include "Column.h"
#include "Editor.h"
#include "Items.h"
void OnFocusChangedGrid1()
{
   if ( IsWindow( m_grid.m_hWnd ) )
   {
      CItems items = m_grid.GetItems();
      COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
      COleVariant vtFocusCell( items.GetItemCell( items.GetFocusItem(),
COleVariant(m_grid.GetFocusColumnIndex())));
      if ( m_grid.GetSelectColumnInner() > 0 )
         vtFocusCell = items.GetInnerCell( vtMissing, vtFocusCell,
COleVariant(m_grid.GetSelectColumnInner()));
      if ( items.GetCellEditorVisible( vtMissing, vtFocusCell ) )
      {
         CEditor editor;
         if ( items.GetHasCellEditor( vtMissing, vtFocusCell ) )
            editor = items.GetCellEditor( vtMissing, vtFocusCell );
         else
         {
            CColumn column( m_grid.GetColumns().GetItem( COleVariant(
m_grid.GetFocusColumnIndex() ) ) );
            editor = column.GetEditor();
         }
         if ( editor.m_lpDispatch != NULL )
            editor.DropDown();
      }
   }
}
```

The following VB.NET sample shows the drop down portion of an editor as soon as a cell is focused:

```
Private Sub AxGrid1_FocusChanged(ByVal sender As Object, ByVal e As System.EventArgs)
Handles AxGrid1.FocusChanged
   With AxGrid1.Items
      Dim focusCell As Object = .ItemCell(.FocusItem, AxGrid1.FocusColumnIndex)
```

```
    If (AxGrid1.SelectColumnInner > 0) Then
        focusCell = .InnerCell(Nothing, focusCell, AxGrid1.SelectColumnInner)
    End If
    If (.CellEditorVisible(, focusCell)) Then
        Dim ed As EXGRIDLib.Editor =
AxGrid1.Columns(AxGrid1.FocusColumnIndex).Editor
        If (.HasCellEditor(, focusCell)) Then
            ed = .CellEditor(, focusCell)
        End If
        ed.DropDown()
    End If
  End With
End Sub
```

The following C# sample shows the drop down portion of an editor as soon as a cell is focused:

```
private void axGrid1_FocusChanged(object sender, EventArgs e)
{
  EXGRIDLib.Items items = axGrid1.Items;
  object focusCell = items.get_ItemCell(items.FocusItem, axGrid1.FocusColumnIndex);
  if ( axGrid1.SelectColumnInner > 0 )
    focusCell = items.get_InnerCell( null, focusCell, axGrid1.SelectColumnInner );
  if ( items.get_CellEditorVisible(null, focusCell ))
  {
    EXGRIDLib.Editor editor = axGrid1.Columns[axGrid1.FocusColumnIndex].Editor;
    if ( items.get_HasCellEditor(null, focusCell ))
      editor = items.get_CellEditor(null, focusCell );
    if ( editor != null )
      editor.DropDown();
  }
}
```

The following VFP sample shows the drop down portion of an editor as soon as a cell is focused:

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

```
with thisform.Grid1.Items
   local ed
   ed = thisform.Grid1.Columns(thisform.Grid1.FocusColumnIndex).Editor
   if ( .HasCellEditor(.FocusItem, thisform.Grid1.FocusColumnIndex) )
      ed = .CellEditor(.FocusItem, thisform.Grid1.FocusColumnIndex)
   endif
   ed.DropDown()
endwith
```

property Editor.DropDownAlignment as AlignmentEnum

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

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

Use the DropDownAlignment property to align the items in the editor's drop-down list. Also the DropDownAlignment property aligns the drop down portion of the editor. Use the Alignment property to align a column. Use the CellHAlignment property to align a cell. The property has effect only for the drop down type editors.

The DropDownAlignment property accepts the following values (dropdown,caption):

- 0 - (right,left)
- 1 - (right,center)
- 2 - (right,right)
- 16 - (center,left)
- 17 - (center,center)
- 18 - (center,right)
- 32 - (left,left)
- 33 - (left,center)
- 34 - (left,right)

where the (center,right) means that the drop down portion is centered, and the captions are aligned to the right

The following VB sample aligns the predefined items to the right (the editor is assigned to the column):

```
With Grid1
  .TreeColumnIndex = -1
  With .Columns.Add("CheckList")
  .Alignment = RightAlignment
    With .Editor
      .EditType = CheckListType
      .DropDownAlignment = RightAlignment
      .AddItem &H1, "ReadOnly", 1
      .AddItem &H2, "Hidden", 2
      .AddItem &H4, "System", 3
```

```
        .AddItem &H10, "Directory", 4
        .AddItem &H20, "Archive", 5
        .AddItem &H80, "Normal", 7
        .AddItem &H100, "Temporary", 8
      End With
    End With
    .Items.AddItem &H1 + &H2
End With
```

In the above sample, the TreeColumnIndex is set to -1, because the Alignment property is not applied for column that displays the hierarchy.

The following VB sample adds an editor that aligns its predefined items to the right:

```
With Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .DropDownAlignment = RightAlignment
      .EditType = DropDownListType
      .AddItem 0, "No border"
      .AddItem 1, "Single Border"
      .AddItem 2, "Double Border"
   End With
End With
```

The following C++ sample adds an editor that aligns its predefined items to the right:

```
#include "Items.h"
#include "Editor.h"
COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
CItems items = m_grid.GetItems();
CEditor editor = items.GetCellEditor( COleVariant( items.GetFirstVisibleItem() ),
COleVariant( long(0) ) );
editor.SetEditType( 3 /*DropDownList*/ );
editor.SetDropDownAlignment( 2 /*RightAlignment*/ );
editor.AddItem( 0, "No border", vtMissing );
editor.AddItem( 1, "Single border", vtMissing );
editor.AddItem( 2, "Double border", vtMissing );
```

The following VB.NET sample adds an editor that aligns its predefined items to the right:

```
With AxGrid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .DropDownAlignment = EXGRIDLib.AlignmentEnum.RightAlignment
      .EditType = EXGRIDLib.EditTypeEnum.DropDownListType
      .AddItem(0, "No border")
      .AddItem(1, "Single Border")
      .AddItem(2, "Double Border")
   End With
End With
```

The following C# sample adds an editor that aligns its predefined items to the right:

```
EXGRIDLib.Items items = axGrid1.Items;
EXGRIDLib.Editor editor = items.get_CellEditor(items.FirstVisibleItem, 0);
editor.EditType = EXGRIDLib.EditTypeEnum.DropDownListType ;
editor.DropDownAlignment = EXGRIDLib.AlignmentEnum.RightAlignment;
editor.AddItem(0, "No border", null);
editor.AddItem(1, "Single border", null);
editor.AddItem(2, "Double border", null);
```

The following VFP sample adds an editor that aligns its predefined items to the right:

```
with thisform.Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = 3 && DropDownList
      .DropDownAlignment = 2 && RightAlignment
      .AddItem(0, "No border")
      .AddItem(1, "Single Border")
      .AddItem(2, "Double Border")
   EndWith
endwith
```

property Editor.DropDownAutoWidth as DropDownWidthType

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

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

Use the DropDownAutoWidth property to specify when you let the control computes the drop-down list width. The DropDownMinWidth property specifies the minimum width for the drop down portion of the editor. By default, the DropDownAutoWidth property is exDropDownAutoWidth. Use the DropDown method to programmatically show the drop down portion of an editor. Use the DropDownRows property to specify the number of visible rows in the drop down portion of the control.

property Editor.DropDownMinWidth as Long

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

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

The DropDownMinWidth property has no effect if the DropDownAutoWidth property is True. Use the DropDown method to programmatically show the drop down portion of an editor. Use the DropDownRows property to specify the number of visible rows in the drop down portion of the control.

property Editor.DropDownRows as Long

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

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

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

property Editor.DropDownVisible as Boolean

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

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

Use the DropDownVisible property to hide the editor's drop-down button. Use the ButtonWidth property to hide the editor buttons. Use the AddItem, InsertItem method to add predefined values to the drop down list. Use the Refresh method update immediately the cell's content when adding new items to a drop down list editor. If the drop down button is hidden, the editor can't open its drop down portion if the user double clicks the editor, or presses the F4 key.

The following VB sample to check whether the editor's drop down portion is visible:

```
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal
lpClassName As String, ByVal lpWindowName As String) As Long

Private Function isDropped()
   ' Specifies whether the control's drop down portion is visible or not
   isDropped = Not FindWindow("HostPopupWindow", "") = 0
End Function
```

The following VB sample advance to the next line when the ENTER key is pressed, and does the default action when an editor of drop down type is opened:

```
 Private Sub Grid1_KeyDown(KeyCode As Integer, Shift As Integer)
   If (KeyCode = 13) Then
      If Not isDropped() Then
         KeyCode = vbKeyDown
      End If
   End If
End Sub
```

The following VB sample hides the drop down button:

```
With Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
```

```
    .EditType = EXGRIDLibCtl.DropDownListType
    .AddItem 0, "0 - <b>No</b>", 1
    .AddItem 1, "1 - <b>Yes</b>", 2
    .DropDownVisible = False
  End With
  .CellValue(.FirstVisibleItem, 0) = 1
  .CellValueFormat(.FirstVisibleItem, 0) = exHTML
End With
```

The following C++ sample hides the drop down button:

```
#include "Items.h"
#include "Editor.h"
COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
CItems items = m_grid.GetItems();
COleVariant vtItem( items.GetFirstVisibleItem() ), vtColumn( long(0) );
CEditor editor = items.GetCellEditor( vtItem, vtColumn );
editor.SetEditType( 3 /*DropDownListType*/ );
editor.AddItem( 0, "0 - <b>No</b>", vtMissing );
editor.AddItem( 1, "1- <b>Yes</b>", vtMissing );
editor.SetDropDownVisible( FALSE );
items.SetCellValue( vtItem, vtColumn, COleVariant( long(1)) );
items.SetCellValueFormat( vtItem, vtColumn, 1 /*exHTML*/ );
```

The following VB.NET sample hides the drop down button:

```
With AxGrid1.Items
  With .CellEditor(.FirstVisibleItem, 0)
    .EditType = EXGRIDLib.EditTypeEnum.DropDownListType
    .AddItem(0, "0 - <b>No</b>", 1)
    .AddItem(1, "1 - <b>Yes</b>", 2)
    .DropDownVisible = False
  End With
  .CellValue(.FirstVisibleItem, 0) = 1
  .CellValueFormat(.FirstVisibleItem, 0) = EXGRIDLib.ValueFormatEnum.exHTML
End With
```

The following C# sample hides the drop down button:

```
EXGRIDLib.Items items = axGrid1.Items;
EXGRIDLib.Editor editor = items.get_CellEditor(items.FirstVisibleItem, 0);
editor.EditType = EXGRIDLib.EditTypeEnum.DropDownListType ;
editor.AddItem(0, "0 - <b>No</b>", 1);
editor.AddItem(1, "1 - <b>Yes</b>", 1);
editor.DropDownVisible = false;
items.set_CellValue( items.FirstVisibleItem, 0, 1 );
items.set_CellValueFormat(items.FirstVisibleItem, 0, EXGRIDLib.ValueFormatEnum.exHTML);
```

The following VFP sample hides the drop down button:

```
with thisform.Grid1.Items
   .DefaultItem = .FirstVisibleItem
   With .CellEditor(0, 0)
      .EditType = 3 && DropDownListType
      .AddItem(0, "0 - <b>No</b>", 1)
      .AddItem(1, "1 - <b>Yes</b>", 2)
      .DropDownVisible = .f.
   EndWith
   .CellValue(0,0) = 1
   .CellValueFormat(0,0) = 1
endwith
```

property Editor.EditType as EditTypeEnum

Specifies the type of the column's editor.

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

Use the EditType property to set the type of the editor. By default, the editor's type is ReadOnly. If the control is bound to an ADO recordset the control looks for appropriate editor for each field. Each column has its own editor, that means that all cells of the column will use the column's editor when users edits a cell. The editor is visible only if the CellEditorVisible property is True. If the EditType is UserEditorType the UserEditor property should be called to initialize the user editor based on the ActiveX's identifier. Use the CellEditor property to specify a particular editor for a specific cell. Use the Option property to define options for a specific type of editor. Use the DefaultEditorOption property to specify default option for the editors of a specified type. Use the Locked property to lock an editor.

The following VB sample sets the column's editor to CheckListType:

```
Set c = .Columns.Add("Description")
With c.Editor
   .EditType = CheckListType
   .AddItem &H1000, "adFldCacheDeferred", 3
   .AddItem &H10, "adFldFixed", 3
   .AddItem &H40000, "adFldIsCollection", 3
   .AddItem &H20000, "adFldIsDefaultStream", 3
   .AddItem &H20, "adFldIsNullable", 3
   .AddItem &H10000, "adFldIsRowURL", 3
   .AddItem &H80, "adFldLong", 3
   .AddItem &H40, "adFldMayBeNull", 3
   .AddItem &H2, "adFldMayDefer", 3
   .AddItem &H4000, "adFldNegativeScale", 3
   .AddItem &H100, "adFldRowID", 3
   .AddItem &H200, "adFldRowVersion", 3
   .AddItem &H8, "adFldUnknownUpdatable", 3
   .AddItem &H4, "adFldUpdatable", 3
End With
```

In this case, the value of CellValue property for any cell in "Description" column should be an OR combination of values added using AddItem, InsertItem methods.

The following VB sample adds an EditType editor to the first visible item:

```
With Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = EXGRIDLibCtl.EditType
   End With
End With
```

The following C++ sample adds an EditType editor to the first visible item:

```
#include "Items.h"
#include "Editor.h"
COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
CItems items = m_grid.GetItems();
CEditor editor = items.GetCellEditor( COleVariant( items.GetFirstVisibleItem() ),
COleVariant( long(0) ) );
editor.SetEditType( 1 /*EditType*/ );
```

The following VB.NET sample adds an EditType editor to the first visible item:

```
With AxGrid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = EXGRIDLib.EditTypeEnum.EditType
   End With
End With
```

The following C# sample adds an EditType editor to the first visible item:

```
EXGRIDLib.Items items = axGrid1.Items;
EXGRIDLib.Editor editor = items.get_CellEditor(items.FirstVisibleItem, 0);
editor.EditType = EXGRIDLib.EditTypeEnum.EditType ;
```

The following VFP sample adds an EditType editor to the first visible item:

```
with thisform.Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = 1 && EdiType
```

```
    EndWith
endwith
```

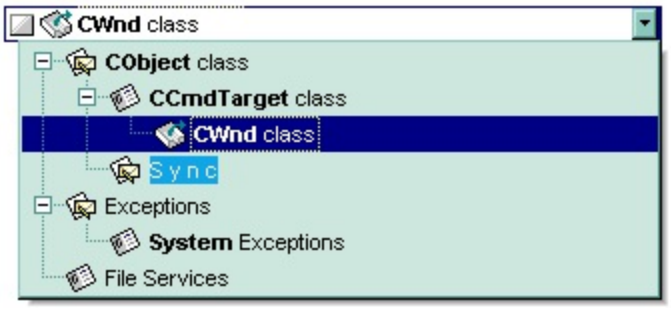
method Editor.ExpandAll ()

Expands all items in the editor's list.

Type	Description

By default, in your editor items that contain child items are collapsed. Use the ExpandAll method to expand all items in the editor. Use the InsertItem method to insert child items. Use the AddItem method to add predefined values to a drop down type editor. Use the ExpandItem method to expand a predefined value. Call the ExpandAll method after inserting the items. Use the SortItems method to sort the items in a drop down editor.

The following screen show shows a simple hierarchy into a built-in DropDownList editor:



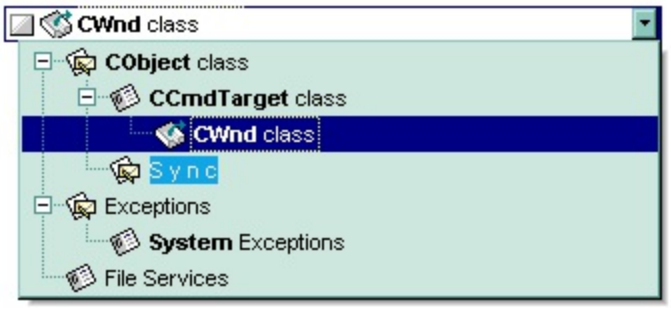
property Editor.ExpandItem(Value as Variant) as Boolean

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

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

By default, the items in a drop down editor are collapsed. Use the ExpandItem to expand a specified item. Use the ExpandAll method to expand all items in the editor. Use the InsertItem method to insert a child item to your built-in editor. Use the AddItem method to add new predefined values. Use the AddButton method to add a button to the editor. Use the DropDownAutoWidth property on False, when inserting predefined values as child items.

The following screen show shows a simple hierarchy into a built-in DropDownList editor:



The following VB sample adds a simple hierarchy to a DropDownList built-in editor:

```
Dim h1 As HITEM
With Grid1
   .BeginUpdate
   .Images
("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUlk8plUrlktl0vmEx

   .Images
("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUlk8plUrlktl0vmEx

   .Images
("gBJJgBggAAkGAAQhIAf8Nf4hhkOiRCJo2AEXjAAi0XFEYIEYhUXAIAEEZi8hk0plUrlktl0vmEx
```

```
    .Images
("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUlk8plUrlktl0vmExı

    .Columns.Add "Column 1"
    .ColumnAutoResize = True
    .HeaderVisible = False
    With .Items
        h1 = .InsertItem(, , "Child 1") ' Inserts a child itme
        .CellValueFormat(h1, 0) = exHTML
        .CellHasCheckBox(h1, 0) = True   ' Associates a check box to a cell
        .CellValue(h1, 0) = 3            ' Sets the cell's value
        .CellImage(h1, 0) = 1            ' Associates an image to a cell
        With .CellEditor(h1, 0)
            .EditType = DropDownListType
            .DropDownAutoWidth = False
            .AddItem 1, "CObject class", 1
            .AddItem 2, "CCmdTarget class", 2, 1
            .AddItem 3, "CWnd class", 3, 2
            .AddItem 6, "S y n c", 1, 1
            .AddItem 4, "Exceptions", 1
            .AddItem 7, "System Exceptions", 2, 4
            .AddItem 5, "File Services", 2
            .ExpandAll
        End With
    End With
    .EndUpdate
End With
```

The following VB samples adds a simple hierarchy to a PickEditType buil-in editor:

```
With Grid1.Items
    With .CellEditor(.FirstVisibleItem, 0)
        .EditType = EXGRIDLibCtl.PickEditType
        .AddItem 0, "Organization"
        .InsertItem 1, "UN", , 0
        .InsertItem 2, "ONU", , 0
        .ExpandItem(0) = True
```

```
    End With
End With
```

The following C++ sample adds a simple hierarchy to a PickEditType built-in editor:

```
#include "Items.h"
#include "Editor.h"
COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
CItems items = m_grid.GetItems();
CEditor editor = items.GetCellEditor( COleVariant( items.GetFirstVisibleItem() ),
COleVariant( long(0) ) );
editor.SetEditType( 14 /*PickEditType*/ );
editor.AddItem( 0, "Organization", vtMissing );
editor.InsertItem( 1, "UN", vtMissing, COleVariant( long(0) ) );
editor.InsertItem( 2, "ONU", vtMissing, COleVariant( long(0) ) );
editor.SetExpandItem( COleVariant(long(0)), TRUE);
```

The following VB.NET sample adds a simple hierarchy to a DropDownListType built-in editor:

```
With AxGrid1.Items
    With .CellEditor(.FirstVisibleItem, 0)
        .EditType = EXGRIDLib.EditTypeEnum.DropDownListType
        .AddItem(0, "Organization")
        .InsertItem(1, "UN", , 0)
        .InsertItem(2, "ONU", , 0)
        .ExpandItem(0) = True
    End With
End With
```

The following C# sample adds a simple hierarchy to a DropDownListType built-in editor:

```
EXGRIDLib.Items items = axGrid1.Items;
EXGRIDLib.Editor editor = items.get_CellEditor(items.FirstVisibleItem, 0);
editor.EditType = EXGRIDLib.EditTypeEnum.DropDownListType;
editor.AddItem(0, "Organization", null);
editor.InsertItem(1, "UN",null,  0);
editor.InsertItem(2, "ONU",null, 0);
editor.set_ExpandItem(0, true);
```

The following VFP sample adds a simple hierarchy to a DropDownType built-in editor:

```
with thisform.Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = 14 && PickEdiType
      .AddItem(0, "Organization")
      .InsertItem(1, "UN", , 0)
      .InsertItem(2, "ONU", , 0)
      .ExpandItem(0) = .t.
   EndWith
endwith
```

property Editor.FindItem (Value as Variant) as Variant

Finds an item given its value or caption.

Type	Description
Value as Variant	A long expression that indicates the value of the item being searched, a string expression that indicates the caption of the item being searched. If searching for a caption (string parameter), it can starts with ">" character, and so the searching is case-insensitive. By default the searching is case-sensitive. For instance, FindItem("One") looks for the exactly caption "One", while if using as FindItem(">One"), it searches case-insensitive for the word "one".
Variant	A string expression that indicates the caption of the item, if the Value is a long expression, a long expression that indicates the item's value if Value is a string expression.

The FindItem property retrieves an empty (VT_EMPTY) value if no item was found. Use the FindItem property to search the caption of the predefined value, in case the Value parameter is a long expression, or look for the predefined value when the Value parameter is a string expression. Use the AddItem method to add predefined values. Use the InsertItem method to add predefined values as child items.

In case you are using Items.CellEditor, the following sample finds the caption of selected value within a cell's editor. For instance, the NewValue can be the NewValue parameter of the Change event:

```
With Grid1
  With .Items
    If (.HasCellEditor(Item, ColIndex)) Then
      Debug.Print ("Caption: " & .CellEditor(Item, ColIndex).FindItem(NewValue))
    End If
  End With
End With
```

In case you are using Column.Editor, the following sample finds the caption of selected value within a column's editor. For instance, the NewValue can be the NewValue parameter of the Change event:

```
With Grid1
  With .Columns
```

```
        Debug.Print ("Caption: " & .Item(ColIndex).Editor.FindItem(NewValue))
    End With
End With
```

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

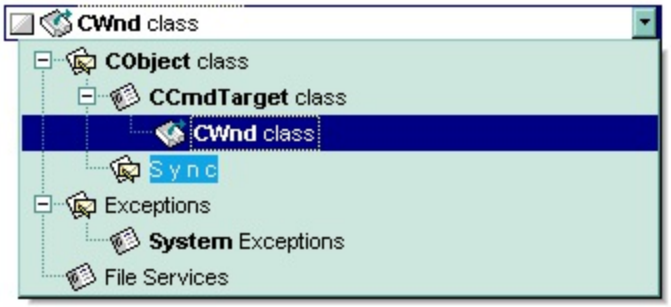
Inserts a child item to the editor's list.

Type	Description
Value as Long	A long expression that defines an unique predefined value
Caption as String	A string expression that specifies the HTML caption associated with the value. The format of the Caption parameter is "key|captionŚcaptionŚ...Ścaption", which indicates an item with the giving key / identifier, which displays multiple captions. The Caption allows using the following special characters: • | character (pipe, vertical bar, ALT + 126) defines the key or identifier of the item to add. Currently, the key is used by a **DropDownListType** editor to specify string codes rather numeric values for the cell's value (CellValue property) • Ś character (vertical broken bar, ALT + 221) defines captions for multiple columns. The Ś character can be escaped, so \Ś displays the Ś character. (available for DropDownType, DropDownListType and PickEditType editors, 20.0+) For instance: • "New York City" defines the "**New York** City" item • "NYC|New York City" the "**New York** City" item with the "NYC" as key or identifier • "NYC|New York CityŚ783.8 km¸Ś8.42 mil" defines the "**New York** City" item with the "NYC" as key or identifier and sub-captions 783.8 km¸ and 8.42 mil (in separated columns) • "New York CityŚ783.8 km¸Ś8.42 mil" defines the "**New York** City" item and sub-captions 783.8 km¸ and 8.42 mil (in separated columns)
Image as Variant	A long expression that indicates the index of the item's icon (1-based). Use the Images method to assign a list of icons to the control.

Parent as Variant	A long expression that defines the value of the parent item.

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

The following screen show shows a simple hierarchy into a built-in DropDownList editor:



The following VB sample adds a simple hierarchy to a DropDownList built-in editor:

```
Dim h1 As HITEM
With Grid1
   .BeginUpdate
   .Images
("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUIk8plUrlktl0vmEx

   .Images
("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUIk8plUrlktl0vmEx

   .Images
("gBJJgBggAAkGAAQhIAf8Nf4hhkOiRCJo2AEXjAAi0XFEYIEYhUXAIAEEZi8hk0plUrlktl0vmEx

   .Images
("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUIk8plUrlktl0vmEx

   .Columns.Add "Column 1"
   .ColumnAutoResize = True
   .HeaderVisible = False
   With .Items
      h1 = .InsertItem(, , "Child 1") ' Inserts a child itme
      .CellValueFormat(h1, 0) = exHTML
```

```
        .CellHasCheckBox(h1, 0) = True   ' Associates a check box to a cell
        .CellValue(h1, 0) = 3            ' Sets the cell's value
        .CellImage(h1, 0) = 1            ' Associates an image to a cell
        With .CellEditor(h1, 0)
            .EditType = DropDownListType
            .DropDownAutoWidth = False
            .AddItem 1, "CObject class", 1
            .InsertItem 2, "CCmdTarget class", 2, 1
            .InsertItem 3, "CWnd class", 3, 2
            .InsertItem 6, "S y n c", 1, 1
            .AddItem 4, "Exceptions", 1
            .InsertItem 7, "System Exceptions", 2, 4
            .AddItem 5, "File Services", 2
            .ExpandAll
        End With
    End With
    .EndUpdate
End With
```

The following VB samples adds a simple hierarchy to a PickEditType buil-in editor:

```
With Grid1.Items
    With .CellEditor(.FirstVisibleItem, 0)
        .EditType = EXGRIDLibCtl.PickEditType
        .AddItem 0, "Organization"
        .InsertItem 1, "UN", , 0
        .InsertItem 2, "ONU", , 0
        .ExpandItem(0) = True
    End With
End With
```

The following C++ sample adds a simple hierarchy to a PickEditType built-in editor:

```
#include "Items.h"
#include "Editor.h"
COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
CItems items = m_grid.GetItems();
CEditor editor = items.GetCellEditor( COleVariant( items.GetFirstVisibleItem() ),
```

```
COleVariant( long(0) ) );
editor.SetEditType( 14 /*PickEditType*/ );
editor.AddItem( 0, "Organization", vtMissing );
editor.InsertItem( 1, "UN", vtMissing, COleVariant( long(0) ) );
editor.InsertItem( 2, "ONU", vtMissing, COleVariant( long(0) ) );
editor.SetExpandItem( COleVariant(long(0)), TRUE);
```

The following VB.NET sample adds a simple hierarchy to a DropDownListType built-in editor:

```
With AxGrid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = EXGRIDLib.EditTypeEnum.DropDownListType
      .AddItem(0, "Organization")
      .InsertItem(1, "UN", , 0)
      .InsertItem(2, "ONU", , 0)
      .ExpandItem(0) = True
   End With
End With
```

The following C# sample adds a simple hierarchy to a DropDownListType built-in editor:

```
EXGRIDLib.Items items = axGrid1.Items;
EXGRIDLib.Editor editor = items.get_CellEditor(items.FirstVisibleItem, 0);
editor.EditType = EXGRIDLib.EditTypeEnum.DropDownListType;
editor.AddItem(0, "Organization", null);
editor.InsertItem(1, "UN",null,  0);
editor.InsertItem(2, "ONU",null, 0);
editor.set_ExpandItem(0, true);
```

The following VFP sample adds a simple hierarchy to a DropDownType built-in editor:

```
with thisform.Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = 14 && PickEdiType
      .AddItem(0, "Organization")
      .InsertItem(1, "UN", , 0)
      .InsertItem(2, "ONU", , 0)
      .ExpandItem(0) = .t.
```

```
    EndWith
endwith
```

property Editor.ItemToolTip(Value as Variant) as String

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

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

Use the ItemToolTip property to assign a tooltip for a drop down list value. Use the AddItem or InsertItem methods to insert new items to the drop down predefined list. The ItemToolTip property may include HTML tags that are listed here here.

The following VB sample adds a predefined value that displays a tooltip when the cursor hovers the value in the drop down portion of the editor:

```
With Grid1.Columns(0).Editor
   .EditType = DropDownListType
   .AddItem 1, "Root Item"
   .ItemToolTip(1) = "This is a bit of text that should appear when the cursor
<b>hovers</b> the item."
   .InsertItem 2, "Child Item", , 1
End With
```

The following C++ sample adds a predefined value that displays a tooltip when the cursor hovers the value in the drop down portion of the editor:

```
#include "Items.h"
#include "Editor.h"
COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
CItems items = m_grid.GetItems();
CEditor editor = items.GetCellEditor( COleVariant( items.GetFirstVisibleItem() ),
COleVariant( long(0) ) );
editor.SetEditType( 3 /*DropDownListType*/ );
editor.AddItem( 0, "tooltip", vtMissing );
editor.SetItemToolTip( COleVariant( "tooltip" ), "This is a bit of text that should appear
```

```
when cursor hovers the item.");
```

The following VB.NET sample adds a predefined value that displays a tooltip when the cursor hovers the value in the drop down portion of the editor:

```
With AxGrid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = EXGRIDLib.EditTypeEnum.DropDownListType
      .AddItem(0, "tooltip")
      .ItemToolTip(0) = "This is a bit of text that should appear when cursor hovers the
item."
   End With
End With
```

The following C# sample adds a predefined value that displays a tooltip when the cursor hovers the value in the drop down portion of the editor:

```
EXGRIDLib.Items items = axGrid1.Items;
EXGRIDLib.Editor editor = items.get_CellEditor(items.FirstVisibleItem, 0);
editor.EditType = EXGRIDLib.EditTypeEnum.DropDownListType;
editor.AddItem(0, "tooltip", null);
editor.set_ItemToolTip(0, "This is a bit of text that should appear when cursor hovers the
item.");
```

The following VFP sample adds a predefined value that displays a tooltip when the cursor hovers the value in the drop down portion of the editor:

```
with thisform.Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = 3 && DropDownListType
      .AddItem(0, "tooltip")
      .ItemToolTip(0) = "This is a bit of text that should appear when cursor hovers the
item."
   EndWith
endwith
```

property Editor.Locked as Boolean

Determines whether the editor is locked or unlocked.

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

Use the Locked property to lock the editor. If the editor is locked, the user is not able to change the control's content using the editor. Use the CellEditorVisible property to hide the cell's editor. For instance, if the editor displays a drop down portion, even if locked, it is visible, but the user can't select new items to change the cell's value. Use the ReadOnly property to make the control read only. If the ReadOnly property is exLocked, all editors are locked. If the editor is locked, the user still can use the editor's buttons. The control fires the ButtonClick event when the user clicks a button. Use the Option(exEditLockedBackColor) and Option(exEditLockedForeColor) property to specify background and foreground colors while the edit control is locked.

property Editor.Mask as String

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

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

Use the Mask property to filter characters during data input. Use the Mask property to control the entry of many types of formatted information such as telephone numbers, social security numbers, IP addresses, license keys etc. The Mask property has effect for the following edit types: DropDownType, SpinType, DateType, MaskType, FontType, PickEditType. The Numeric property specifies whether the editor enables numeric values only. Use the MaskChar property to change the masking character. The Mask property is composed by a combination of regular characters, literal escape characters, and placeholders, masking characters. The Mask property can contain also alternative characters, or range rules.

Starting from the version **8.0**, the Mask property has been changed radically, to support more special characters, validation, float numbers, and so on.

For instance, the following input-mask (ext-phone)

"*!(999) 000 0000;1;;select=1,empty,overtype,warning=invalid character,invalid=The value you entered isn't appropriate for the input mask '<%mask%>' specified for this field.*"

indicates the following:

- The pattern should contain 3 optional digits *999*, and 7 required digits *000 0000*, aligned to the right, *!*.
- The second part of the input mask indicates *1*, which means that all literals are included when the user leaves the field.
- The entire field is selected when it receives the focus, *select=1*
- The field supports *empty* value, so the user can leave the field with no content
- The field enters in *overtype* mode, and insert-type mode is not allowed when user pressed the Insert key
- If the user enters any invalid character, a *warning* tooltip with the message "*invalid character*" is displayed.
- If the user tries to leave the field, while the field is not validated (all 7 required digits completed), the *invalid* tooltip is shown with the message "*The value you entered isn't*

appropriate for the input mask '<%mask%>' specified for this field." The *<%mask%>* is replaced with the first part of the input mask *!(999) 000 0000*

The four parts of an input mask, or the Mask property supports up to four parts, separated by a semicolon (;). For instance, "`Time: `00:00:00;;0;overtype,warning=<fgcolor FF0000>invalid character,beep", indicates the pattern "00:00" with the prefix Time:, the masking character being the 0, instead _, the field enters in over-type mode, insert-type mode is not allowed, and the field beeps and displays a tooltip in red with the message invalid character when the user enters an invalid character.

Input masks are made up one mandatory part and three optional parts, and each part is separated by a semicolon (;). If a part should use the semicolon (;) it must uses the \; instead

The purpose of each part is as follows:

1. The first part (pattern) is mandatory. It includes the mask characters or string (series of characters) along with placeholders and literal data such as, parentheses, periods, and hyphens.

   The following table lists the placeholder and literal characters for an input mask and explains how it controls data entry:

   - ***#**, a digit, +, - or space (entry not required).*
   - ***0**, a digit (0 through 9, entry required; plus [+] and minus [-] signs not allowed).*
   - ***9**, a digit or space (entry not required; plus and minus signs not allowed).*
   - ***x**, a lower case hexa character, [0-9],[a-f] (entry required)*
   - ***X**, an upper case hexa character, [0-9],[A-F] (entry required)*
   - ***A**, any letter, digit (entry required).*
   - ***a**, any letter, digit or space (entry optional).*
   - ***L**, any letter (entry require).*
   - ***?**, any letter or space (entry optional).*
   - ***&**, any character or a space (entry required).*
   - ***C**, any character or a space (entry optional).*
   - ***>**, any letter, converted to uppercase (entry required).*
   - ***<**, any letter, converted to lowercase (entry required).*
   - ****, any characters combinations***
   - ***{ min,max }** (Range), indicates a number range. The syntax {min,max} (Range), masks a number in the giving range. The min and max values should be positive integers. For instance the mask {0,255} masks any number between 0 and 255.*
   - ***[...]** (Alternative), masks any characters that are contained in the [] brackets. For*

instance, the [abcdA-D] mask any character: a,b,c,d,A,B,C,D

- *******, indicates the escape character*
- ***t'****, (ALT + 175) causes the characters that follow to be converted to uppercase, until Ť(ALT + 174) is found.*
- ***Ť****, (ALT + 174) causes the characters that follow to be converted to lowercase, until t'(ALT + 175) is found.*
- ***!****, causes the input mask to fill from right to left instead of from left to right.*

Characters enclosed in double quotation ("" or ``) marks will be displayed literally. If this part should display/use the semicolon (;) character is should be included between double quotation ("" or ``) characters or as \; (escape).

2. The second part is optional and refers to the embedded mask characters and how they are stored within the field. If the second part is set to 0 (default, exClipModeLiteralsNone), all characters are stored with the data, and if it is set to 1 (exClipModeLiteralsInclude), the literals are stored, not including the masking/placeholder characters, if 2 (exClipModeLiteralsExclude), just typed characters are stored, if 3(exClipModeLiteralsEscape), optional, required, editable and escaped entities are included. No double quoted text is included.
3. The third part of the input mask is also optional and indicates a single character or space that is used as a placeholder. By default, the field uses the underscore (_). If you want to use another character, enter it in the third part of your mask. Only the first character is considered. If this part should display/use the semicolon (;) character is should be \; (escape) (MaskChar property)
4. The forth part of the input, indicates a list of options that can be applied to input mask, separated by comma(,) character.

   The known options for the forth part are:

- ***float****, indicates that the field is edited as a decimal number, integer. The first part of the input mask specifies the pattern to be used for grouping and decimal separators, and - if negative numbers are supported. If the first part is empty, the float is formatted as indicated by current regional settings. For instance, "##;;;float" specifies a 2 digit number in float format. The grouping, decimal, negative and digits options are valid if the float option is present.*
- ***grouping****=value, Character used to separate groups of digits to the left of the decimal. Valid only if float is present. For instance ";;;float,grouping=" indicates that no grouping is applied to the decimal number (LOCALE_STHOUSAND)*
- ***decimal****=value, Character used for the decimal separator. Valid only if float is*

present. For instance ";;;float,grouping= ,decimal=\," indicates that the decimal number uses the space for grouping digits to the left, while for decimal separator the comma character is used (LOCALE_SDECIMAL)

- ***negative**=value, indicates whether the decimal number supports negative numbers. The value should be 0 or 1. 1 means negative numbers are allowed. Else 0 or missing, the negative numbers are not accepted. Valid only if float is present.*
- ***digits**=value, indicates the max number of fractional digits placed after the decimal separator. Valid only if float is present. For instance, ";;;float,digits=4" indicates a max 4 digits after decimal separator (LOCALE_IDIGITS)*
- ***password**[=value], displays a black circle for any shown character. For instance, ";;;password", specifies that the field to be displayed as a password. If the value parameter is present, the first character in the value indicates the password character to be used. By default, the * password character is used for non-TrueType fonts, else the black circle character is used. For instance, ";;;password=*", specifies that the field to be displayed as a password, and use the * for password character. If the value parameter is missing, the default password character is used.*
- ***right**, aligns the characters to the right. For instance, "(999) 999-9999;;;right" displays and masks a telephone number aligned to the right*
- ***readonly**, the editor is locked, user can not update the content, the caret is available, so user can copy the text, excepts the password fields.*
- ***inserttype**, indicates that the field enters in insert-type mode, if this is the first option found. If the forth part includes also the overtype option, it indicates that the user can toggle the insert/over-type mode using the Insert key. For instance, the "##:##;;0;inserttype,overtype", indicates that the field enter in insert-type mode, and over-type mode is allowed. The "##:##;;0;inserttype", indicates that the field enter in insert-type mode, and over-type mode is not allowed.*
- ***overtype**, indicates that the field enters in over-type mode, if this is the first option found. If the forth part includes also the inserttype option, it indicates that the user can toggle the insert/over-type mode using the Insert key. For instance, the "##:##;;0;overtype,inserttype", indicates that the field enter in over-type mode, and insert-type mode is allowed. The "##:##;;0;overtype", indicates that the field enter in over-type mode, and insert-type mode is not allowed.*
- ***nocontext**, indicates that the field provides no context menu when user right clicks the field. For instance, ";;;password,nocontext" displays a password field, where the user can not invoke the default context menu, usually when a right click occurs.*
- ***beep**, indicates whether a beep is played once the user enters an invalid character. For instance, "00:00;;;beep" plays a beep once the user types in invalid character, in this case any character that's not a digit.*

- ***warning***=*value, indicates the html message to be shown when the user enters an invalid character. For instance, "00:00:00;;;warning=invalid character" displays a "invalid character" tooltip once the user types in invalid character, in this case any character that's not a digit. The <%mask%> keyword in value, substitute the current mask of the field, while the <%value%> keyword substitutes the current value (including the literals). If this option should display/use the semicolon (;) character is should be \; (escape)*
- ***invalid***=*value, indicates the html message to be displayed when the user enters an inappropriate value for the field. If the value is missing or empty, the option has no effect, so no validation is performed. If the value is a not-empty value, the validation is performed. If the value is single space, no message is displayed and the field is keep opened while the value is inappropriate. For instance, "!(999) 000 0000;;;invalid=The value you entered isn't appropriate for the input mask '<%mask%>' specified for this field." displays the "The value you entered isn't appropriate for the input mask '...' specified for this field." tooltip once the user leaves the field and it is not-valid (for instance, the field includes entities required and uncompleted). The <%mask%> keyword in value, substitute the current mask of the field, while the <%value%> keyword substitutes the current value (including the literals). If this option should display/use the semicolon (;) character is should be \; (escape). This option can be combined with empty, validateas. The invalid option should be used, with the CauseValidateValue property on True, so the user can not leaves the field while it contains an invalid value.*
- ***validateas***=*value, specifies the additional validation is done for the current field. If value is missing or 0 (exValidateAsNone), the option has no effect. The validateas option has effect only if the invalid option specifies a not-empty value. Currently, the value can be 1 (exValidateAsDate), which indicates that the field is validated as a date. For instance, having the mask "!00/00/0000;;0;empty,validateas=1,invalid=Invalid date!,warning=Invalid character!,select=4,overtype", indicates that the field is validate as date (validateas=1).*
- ***empty****, indicates whether the field supports empty values. This option can be used with invalid flag, which indicates that the user can leave the field if it is empty. If empty flag is present, the field displays nothing if no entity is completed (empty). Once the user starts typing characters the current mask is displayed. For instance, having the mask "!(999) 000 0000;;;empty,select=4,overtype,invalid=invalid phone number,beep", it specifies an empty or valid phone to be entered.*
- ***select***=*value, indicates what to select from the field when it got the focus. The value could be 0 (nothing, exSelectNoGotFocus), 1 (select all, exSelectAllGotFocus), 2 (select the first empty and editable entity of the field,*

exSelectEditableGotFocus), 3 (moves the cursor to the beginning of the first empty and editable entity of the field, exMoveEditableGotFocus), 4 (select the first empty, required and editable entity of the field, exSelectRequiredEditableGotFocus), 5 (moves the cursor to the beginning of the first empty, required and editable entity of the field, exMoveRequiredEditableGotFocus). For modes 2 and 4 the entire field is selected if no matching entity is found. For instance, "`Time:`XX:XX;;;select=1" indicates that the entire field (including the Time: prefix) is selected once it get the focus. The "`Time:`XX:XX;;;select=3", moves the cursor to first X, if empty, the second if empty, and so on

- ***leading**=value, specifies whether the spaces or masking/placeholder (0,9) characters are replaced with giving value (0 if the value is missing). This option has effect, only for DateType fields (Editor.EditType property is DateType), when the field is entering in edit mode, or the user selects a new date from the drop down calendar. For instance, "!99/99/9999;1;;empty,validateas=1,invalid=Invalid date\, for the input mask
'<%mask%>'!,warning=Invalid character!,select=4,overtype,leading", having the cell's value on #1/1/2001# it displays 01/01/2001, instead 1/1/2001.*

Experimental:
***multiline**, specifies that the field supports multiple lines.*
***rich**, specifies that the field displays a rich type editor. By default, the standard edit field is shown*
***disabled**, shows as disabled the field.*

The following special characters are supported only in versions prior to version **8.0**

Here's the list of all rules and masking characters:

- **#** (Digit), Masks a digit character. [0-9]
- **x** (Hexa Lower), Masks a lower hexa character. [0-9],[a-f]
- **X** (Hexa Upper), Masks a upper hexa character. [0-9],[A-F]
- **A** (AlphaNumeric), Masks a letter or a digit. [0-9], [a-z], [A-Z]
- **?** (Alphabetic), Masks a letter. [a-z],[A-Z]
- **<** (Alphabetic Lower), Masks a lower letter. [a-z]
- **>** (Alphabetic Upper), Masks an upper letter. [A-Z]
- ***** (Any), Mask any combination of characters.
- **** (Literal Escape), Displays any masking characters. The following combinations are valid: \#,\x,\X,\A,\?,\<,\>,\\,\{,\[
- **{nMin,nMax}** (Range), Masks a number in a range. The nMin and nMax values should

be numbers. For instance the mask {0,255} will mask any number between 0 and 255.

- **[...]** (Alternative), Masks any characters that are contained by brackets []. For instance, the [abcA-C] mask any character: a,b,c,A,B,C

The following VB sample adds a mask for IP addresses:

```
With Grid1
   With .Columns.Add("Mask")
      With .Editor
         .EditType = EditTypeEnum.MaskType
         .Mask = "{0,255}\.{0,255}\.{0,255}\.{0,255}"
      End With
   End With
   .Items.AddItem "193.226.40.161"
End With
```

The following VB sample masks a phone number:

```
With Grid1
   With .Columns.Add("Mask")
      With .Editor
         .EditType = EditTypeEnum.MaskType
         .Mask = "(XXX) - XXX XXXX"
      End With
   End With
   .Items.AddItem "(095) - 889 1234"
End With
```

The following C++ adds a mask editor to filter characters while entering a phone number:

```
#include "Items.h"
#include "Editor.h"
COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
CItems items = m_grid.GetItems();
CEditor editor = items.GetCellEditor( COleVariant( items.GetFirstVisibleItem() ),
COleVariant( long(0) ) );
editor.SetEditType( 8 /*MaskType*/ );
editor.SetMask("(###) ### - ####");
```

The following VB.NET adds a mask editor to filter characters while entering a phone number:

```
With AxGrid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = EXGRIDLib.EditTypeEnum.MaskType
      .Mask = "(###) ### - ####"
   End With
End With
```

The following C# adds a mask editor to filter characters while entering a phone number:

```
EXGRIDLib.Items items = axGrid1.Items;
EXGRIDLib.Editor editor = items.get_CellEditor(items.FirstVisibleItem, 0);
editor.EditType = EXGRIDLib.EditTypeEnum.MaskType;
editor.Mask = "(###) ### - ####";
```

The following VFP adds a mask editor to filter characters while entering a phone number:

```
with thisform.Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = 8 && MaskType
      .Mask = "(###) ### - ####"
   EndWith
endwith
```

property Editor.MaskChar as Long

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

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

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

property Editor.Numeric as NumericEnum

Specifies whether the editor enables numeric values only.

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

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

property Editor.Option(Name as EditorOptionEnum) as Variant

Specifies an option for the editor.

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

The Option property of Editor object provides the ability to add scroll bars to a memo editor using the exMemoHScrollBar and exMemoVScrollBar options. Use the DefaultEditorOption property to specify default option for the editors of a specified type.

For instance, the following VB sample adds both scroll bar to the editor of the first column:

```
With Grid1.Columns(0).Editor
        .Option(exMemoAutoSize) = False     ' Disables auto resizing when user
alters the text
        .Option(exMemoVScrollBar) = True    ' Adds the vertical scroll bar
        .Option(exMemoHScrollBar) = True    ' Adds the horizontal scroll bar
End With
```

The following VB sample adds a cell with a password editor:

```
With Grid1.Items
   Dim h As HITEM
   h = .InsertItem(, , "password")
   With .CellEditor(h, 0)
      .EditType = EXGRIDLibCtl.EditType
       .Option(EXGRIDLibCtl.EditorOptionEnum.exEditPassword) = True
   End With
End With
```

The following VB sample indicates how to let user uses the left, right arrows, home and end keys to move the cursor inside an editor that displays a caret, instead changing the focused cell:

```
With Grid1.Columns(ColIndex).Editor
        .Option(exLeftArrow) = exHandleEditor
        .Option(exRightArrow) = exHandleEditor
```

```
    .Option(exHomeKey) = exHandleEditor
    .Option(exEndKey) = exHandleEditor
End With
```

The following C++ sample adds a password editor:

```
#include "Items.h"
#include "Editor.h"
COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
CItems items = m_grid.GetItems();
CEditor editor = items.GetCellEditor( COleVariant( items.GetFirstVisibleItem() ),
COleVariant( long(0) ) );
editor.SetEditType( 1 /*EditType*/ );
editor.SetOption( 18 /*exEditPassword*/, COleVariant( VARIANT_TRUE ) );
```

The following VB.NET sample adds a password editor:

```
With AxGrid1.Items
  With .CellEditor(.FirstVisibleItem, 0)
    .EditType = EXGRIDLib.EditTypeEnum.EditType
    .Option(EXGRIDLib.EditorOptionEnum.exEditPassword) = True
  End With
End With
```

The following C# sample adds a password editor:

```
EXGRIDLib.Items items = axGrid1.Items;
EXGRIDLib.Editor editor = items.get_CellEditor(items.FirstVisibleItem, 0);
editor.EditType = EXGRIDLib.EditTypeEnum.EditType;
editor.set_Option(EXGRIDLib.EditorOptionEnum.exEditPassword, true );
```

The following VFP sample adds a password editor:

```
with thisform.Grid1.Items
  With .CellEditor(.FirstVisibleItem, 0)
    .EditType = 1 && EditType
    .Option(18) = .t. && exEditPassword
  EndWith
endwith
```

property Editor.PartialCheck as Boolean

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

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

Use the PartialCheck property to allow three-states check boxes into the editor. Use the CellHasCheckBox property to define a check box for the cell. Use the CellState property to specify the cell's state. Use the PartialCheck property to allow partial check feature in a column.

property Editor.PopupAppearance as InplaceAppearanceEnum

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

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

By default, the PopupAppearance property is ShadowApp. Use the PopupAppearance property to change the drop-down window's border style. Use the Appearance property to define the editor's appearance.

method Editor.RemoveButton (Key as Variant)

Removes a button given its key.

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

Use the RemoveButton method to remove a button, given its key. Use the ButtonWidth property to hide the editor buttons. Use the ClearButtons method to remove all buttons. Use the DropDownVisible property to hide the drop down button. You can remove only buttons added using the AddButton method.

method Editor.RemoveItem (Value as Long)

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

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

Use the RemoveItem method to remove an item from the editor's predefined values list. Use the ClearItems method to clear the entire list of editor items. Use the DropDownVisible property to hide the drop down button. You can remove only items that were added using AddItem or InsertItem method. Use the FindItem property to look for a predefined value in the drop down list.

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

Sorts the list of items in the editor.

Type	Description
Ascending as Variant	A boolean expression that indicates the sort order of the items. By default, is the Ascending parameter is True, if it is missing.
Reserved as Variant	Not used. For future use only.

Use the SortItems method to sort the items in a drop down editor. Use the ExpandAll method to expand all items. Call the SortItems method after adding predefined values to the drop down list. Use the AddItem or InsertItem method to add predefined values to the drop down list. Use the SortOrder property to sort a column.

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

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

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

The UserEditor property creates a new type of editor based on the ControlID parameter. The EditType property has effect only if it is UserEditorType. Use the UserEditorObject property to access the newly created object. The UserEditorObject property is nothing if the control wasn't able to create the user editor based on the ControlID. Also, if the user control requires a runtime license key, and the License parameter is empty or doesn't match, the UserEditorObject property is nothing. The control fires the UserEditorOpen event when a ActiveX editor is about to be shown. The control fires the UserEditorClose event when the user editor is hidden. The control fires the UserEditorOleEvent event each time when an user editor fires an event. The setup installs the VB\UserEditor, VC\User.Edit sample that uses the Exontrol ExComboBox Component as a new editor.

The following VB sample adds a new column to an editor of of Exontrol.ComboBox type (exComboBox component):

```
With Grid1
  .BeginUpdate
  With .Columns
    With .Add("Column 0").Editor
      .EditType = EditTypeEnum.UserEditorType
      ' Creates an ExComboBox control, and gets the object created
      .UserEditor "Exontrol.ComboBox", ""
       If Not .UserEditorObject Is Nothing Then
         With .UserEditorObject ' Points to an ExComboBox control
           ' The ExComboBox object
           .BeginUpdate
           .EndUpdate
         End With
```

```
        End If
      End With
    End With
    .EndUpdate
End With
```

The following VB sample adds the Exontrol's eXMaskEdit Component to mask floating point numbers with digit grouping:

```
With Grid1.Items
   Dim h As HITEM
   h = .AddItem(100)
   With .CellEditor(h, 0)
      .EditType = UserEditorType
      .UserEditor "Exontrol.MaskEdit", ""
      With .UserEditorObject()
         .BackColor = vbWhite
         .MaskFloat = True
         .Mask = "-###.###.###,##"
      End With
   End With
End With
```

The following C++ sample adds the Exontrol's eXMaskEdit Component to mask floating point numbers with digit grouping:

```
CItems items = m_grid.GetItems();
long hItem = items.AddItem( COleVariant( (double)100 ) );
CEditor editor = items.GetCellEditor( COleVariant( hItem ), COleVariant( long(0) ) );
editor.SetEditType( 16 /*UserEditorType*/ );
editor.UserEditor( "Exontrol.MaskEdit", "" );
MaskEditLib::IMaskEditPtr spMaskEdit( editor.GetUserEditorObject() );
if ( spMaskEdit != NULL )
{
   spMaskEdit->put_MaskFloat( TRUE );
   spMaskEdit->put_Mask( L"-###.###.###,##" );
   spMaskEdit->put_BackColor( RGB(255,255,255) );
}
```

The sample requires calling the #import <maskedit.dll> to include the type library for the eXMaskEdit component. The #import <maskedit.dll> defines the MaskEditLib namespace used in the sample.

The following VB.NET sample adds the Exontrol's eXMaskEdit Component to mask floating point numbers with digit grouping:

```
With AxGrid1.Items
   Dim h As Integer = .AddItem(1000)
   With .CellEditor(h, 0)
      .EditType = EXGRIDLib.EditTypeEnum.UserEditorType
      .UserEditor("Exontrol.MaskEdit", "")
      With .UserEditorObject()
         .BackColor = ToUInt32(Color.White)
         .MaskFloat = True
         .Mask = "-###.###.###,##"
      End With
   End With
End With
```

where the ToUInt32 function converts a Color expression to an unsigned long expression and may look like follows:

```
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 adds the Exontrol's eXMaskEdit Component to mask floating point numbers with digit grouping:

```
EXGRIDLib.Items items = axGrid1.Items;
int hItem = items.AddItem(100);
EXGRIDLib.Editor editor = items.get_CellEditor(hItem, 0);
editor.EditType = EXGRIDLib.EditTypeEnum.UserEditorType;
editor.UserEditor("Exontrol.MaskEdit", "");
```

```
MaskEditLib.MaskEdit maskEdit = editor.UserEditorObject as MaskEditLib.MaskEdit;
if (maskEdit != null)
{
   maskEdit.BackColor = ToUInt32(Color.White);
   maskEdit.MaskFloat = true;
   maskEdit.Mask = "-###.###.###,##";
}
```

where the MaskEditLib class is defined by adding a new reference to the ExMaskEdit component to your project. The ToUInt32 function converts a Color expression to an OLE_COLOR expression (unsigned long expression), and may look like follows:

```
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 adds the Exontrol's eXMaskEdit Component to mask floating point numbers with digit grouping:

```
With thisform.Grid1.Items
   local h
   h = .AddItem(100)
   With .CellEditor(h, 0)
      .EditType = 16 && UserEditorType
      .UserEditor("Exontrol.MaskEdit", "")
      With .UserEditorObject()
         .BackColor = RGB(255,255,255)
         .MaskFloat = .t.
         .Mask = "-###.###.###,##"
      EndWith
   EndWith
EndWith
```

property Editor.UserEditorObject as Object

Gets the user editor object when EditType is UserEditorType.

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

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

The following VB sample initializes an user editor of EXCOMBOBOXLibCtl.ComboBox:

```
With Grid1
  .BeginUpdate
  .DefaultItemHeight = 21
  .TreeColumnIndex = -1
  .ColumnAutoResize = True
  .MarkSearchColumn = False
  .FullRowSelect = False
  .DrawGridLines = exVLines
  With .Columns
    With .Add("Column 0").Editor
    .EditType = EditTypeEnum.UserEditorType
    ' Creates an ExComboBox control, and gets the object created
    .UserEditor "Exontrol.ComboBox", ""
    If Not .UserEditorObject Is Nothing Then
      With .UserEditorObject ' Points to an ExComboBox control
        ' Loads the ExComboBox object
        .BeginUpdate
        .LinesAtRoot = exGroupLinesAtRoot
        .ColumnAutoResize = True
        .IntegralHeight = True
```

```
            .HeaderVisible = False
            .AllowSizeGrip = True
            .MinHeightList = 100
            .AutoDropDown = True
            .HasButtons = exArrow
            .Indent = 18
            .MarkSearchColumn = False
            .BackColor = Grid1.BackColor
            With .Columns
                With .Add("Column 0")
                End With
                With .Add("Column 1")
                    .Position = 0
                    .Width = 16
                End With
            End With
            With .Items
                Dim h1 As HITEM, h2 As HITEM, h12 As HITEM, i As Long
                For i = 1 To 3
                    h1 = .AddItem("Group " & i)
                    .CellCaption(h1, 1) = i * 4 - 3
                    h12 = .InsertItem(h1, , "Item 1")
                    .CellCaption(h12, 1) = i * 4 - 2
                    h12 = .InsertItem(h1, , "Item 2")
                    .CellCaption(h12, 1) = i * 4 - 1
                    h12 = .InsertItem(h1, , "Item 3")
                    .CellCaption(h12, 1) = i * 4
                    .ExpandItem(h1) = True
                Next
            End With
            .EndUpdate
        End With
    Else
        MsgBox "YOU NEED TO HAVE INSTALLED THE EXCOMBOBOX CONTROL, else you will
not be able to see the UserEditor column"
    End If
End With
```

```
With .Add("Column 1")
   With .Editor
      .EditType = EditTypeEnum.DateType
      .AddItem 10, "Ten"
      .AddItem 20, "Twenty"
    End With
End With
End With
For i = 1 To 11
   With .Items
      Dim h As HITEM
      h = .AddItem(i)
   End With
Next
   .EndUpdate
End With
```

The following VB sample adds the Exontrol's eXMaskEdit Component to mask floating point numbers with digit grouping:

```
With Grid1.Items
   Dim h As HITEM
   h = .AddItem(100)
   With .CellEditor(h, 0)
      .EditType = UserEditorType
      .UserEditor "Exontrol.MaskEdit", ""
      With .UserEditorObject()
         .BackColor = vbWhite
         .MaskFloat = True
         .Mask = "-###.###.###,##"
      End With
   End With
End With
```

The following C++ sample adds the Exontrol's eXMaskEdit Component to mask floating point numbers with digit grouping:

```
CItems items = m_grid.GetItems();
```

```
long hItem = items.AddItem( COleVariant( (double)100 ) );
CEditor editor = items.GetCellEditor( COleVariant( hItem ), COleVariant( long(0) ) );
editor.SetEditType( 16 /*UserEditorType*/ );
editor.UserEditor( "Exontrol.MaskEdit", "" );
MaskEditLib::IMaskEditPtr spMaskEdit( editor.GetUserEditorObject() );
if ( spMaskEdit != NULL )
{
   spMaskEdit->put_MaskFloat( TRUE );
   spMaskEdit->put_Mask( L"-###.###.###,##" );
   spMaskEdit->put_BackColor( RGB(255,255,255) );
}
```

The sample requires calling the #import <maskedit.dll> to include the type library for the eXMaskEdit component. The #import <maskedit.dll> defines the MaskEditLib namespace used in the sample.

The following VB.NET sample adds the Exontrol's eXMaskEdit Component to mask floating point numbers with digit grouping:

```
With AxGrid1.Items
   Dim h As Integer = .AddItem(1000)
   With .CellEditor(h, 0)
      .EditType = EXGRIDLib.EditTypeEnum.UserEditorType
      .UserEditor("Exontrol.MaskEdit", "")
      With .UserEditorObject()
         .BackColor = ToUInt32(Color.White)
         .MaskFloat = True
         .Mask = "-###.###.###,##"
      End With
   End With
End With
```

where the ToUInt32 function converts a Color expression to an unsigned long expression and may look like follows:

```
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 adds the Exontrol's eXMaskEdit Component to mask floating point numbers with digit grouping:

```
EXGRIDLib.Items items = axGrid1.Items;
int hItem = items.AddItem(100);
EXGRIDLib.Editor editor = items.get_CellEditor(hItem, 0);
editor.EditType = EXGRIDLib.EditTypeEnum.UserEditorType;
editor.UserEditor("Exontrol.MaskEdit", "");
MaskEditLib.MaskEdit maskEdit = editor.UserEditorObject as MaskEditLib.MaskEdit;
if (maskEdit != null)
{
    maskEdit.BackColor = ToUInt32(Color.White);
    maskEdit.MaskFloat = true;
    maskEdit.Mask = "-###.###.###,##";
}
```

where the MaskEditLib class is defined by adding a new reference to the ExMaskEdit component to your project. The ToUInt32 function converts a Color expression to an OLE_COLOR expression (unsigned long expression), and may look like follows:

```
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 adds the Exontrol's eXMaskEdit Component to mask floating point numbers with digit grouping:

```
With thisform.Grid1.Items
    local h
    h = .AddItem(100)
```

```
    With .CellEditor(h, 0)
        .EditType = 16 && UserEditorType
        .UserEditor("Exontrol.MaskEdit", "")
        With .UserEditorObject()
            .BackColor = RGB(255,255,255)
            .MaskFloat = .t.
            .Mask = "-###.###.###,##"
        EndWith
    EndWith
EndWith
```

ExDataObject object

Defines the object that contains OLE drag and drop information.

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

method ExDataObject.Clear ()

Deletes the contents of the DataObject object.

Type	Description

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

property ExDataObject.Files as ExDataObjectFiles

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

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

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

method ExDataObject.GetData (Format as Integer)

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

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

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

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

method ExDataObject.GetFormat (Format as Integer)

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

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

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

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

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

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

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

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

ExDataObjectFiles object

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

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

method ExDataObjectFiles.Add (FileName as String)

Adds a filename to the Files collection

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

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

method ExDataObjectFiles.Clear ()

Removes all file names in the collection.

Type	Description

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

property ExDataObjectFiles.Count as Long

Returns the number of file names in the collection.

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

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

property ExDataObjectFiles.Item (Index as Long) as String

Returns a specific file name given its index.

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

method ExDataObjectFiles.Remove (Index as Long)

Removes a specific file name given its index into collection.

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

Use Clear method to remove all filenames.

Grid 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: {101EE60F-7B07-48B0-A13A-F32BAE7DA165}. The object's program identifier is: "Exontrol.Grid". The /COM object module is: "ExGrid.dll"

Exontrols new exGrid control an easy-to-implement grid control, provides swift and robust performance and a wide range of formatting features that distinguish it from other grids. It perfectly combines the exTree features with the very popular exEditors library. The object model is rich, intuitive and flexible. The exGrid loads, edits and displays your hierarchical or tabular data. Here's the list of supported properties and methods:

Name	Description
AllowCopyTemplate	Specifies whether the Shift + Ctrl + Alt + Insert sequence copies the control's content to the clipboard, in template form.
AllowGroupBy	Indicates whether the control supports Group-By view.
AllowSelectNothing	Specifies whether the current selection is erased, once the user clicks outside of the items section.
AllowUndoRedo	Enables or disables the Undo/Redo feature.
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.
AutoEdit	Specifies whether the cell is edited once that it is focused.
AutoSearch	Enables or disables the incremental searching feature.
BackColor	Retrieves or sets a value that indicates the control's background color.
BackColorAlternate	Specifies the background color used to display alternate items in the control.
BackColorHeader	Specifies the header's background color.
BackColorLevelHeader	Specifies the multiple levels header's background color.

BackColorLock	Retrieves or sets a value that indicates the control's background color for the locked area.
BackColorSortBar	Retrieves or sets a value that indicates the sort bar's background color.
BackColorSortBarCaption	Returns or sets a value that indicates the caption's background color in the control's sort bar.
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.
CanRedo	Retrieves a value that indicates whether the control can perform a Redo operation.
CanUndo	Retrieves a value that indicates whether the control can perform an Undo operation.
CauseValidateValue	Returns or sets a value that determines whether the ValidateValue event occurs before the user changes the cell's value.
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.
DefaultEditorOption	Specifies a default option for an editor.
DefaultItemHeight	Retrieves or sets a value that indicates the default item height.
Description	Changes descriptions for control objects.
DetectAddNew	Specifies whether the control detects when a new record is added to the bounded recordset.
DetectDelete	Specifies whether the control detects when a record is deleted from the bounded recordset.
DiscardValidateValue	Cancels the current validation process, and restores back the modified cells.
DrawGridLines	Retrieves or sets a value that indicates whether the grid lines are visible or hidden.
Edit	Edits the focused cell.
EditClose	Closes the current editor.
Editing	Specifies the window's handle of the built-in editor while the control is running in edit mode.
EditingText	Specifies the caption of the editor during editing.
Enabled	Enables or disables the control.
EndBlockUndoRedo	Ends recording the UI operations and adds the undo/redo operations as a block, so they all can be restored at once, if Undo method is performed.
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.
	Retrieves or sets a value that indicates the current's event

EventParam	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 filterbar is automatically resized to fit its description.
FilterBarPrompt	Specifies the caption to be displayed when the filter pattern is missing.
FilterBarPromptColumns	Specifies the list of columns to be used when filtering using the prompt.
FilterBarPromptPattern	Specifies the pattern for the filter prompt.
FilterBarPromptType	Specifies the type of the filter prompt.
FilterBarPromptVisible	Shows or hides the filter prompt.
FilterCriteria	Retrieves or sets the filter criteria.
FilterInclude	Specifies the items being included after the user applies the filter.
FocusColumnIndex	Specifies the index of focused column.
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.
Group	Forces the control to do a regrouping of the columns.
GroupUndoRedoActions	Groups the next to current Undo/Redo Actions in a single block.
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 grid 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.
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 the grid'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 the control's image list at runtime.
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.
IsGrouping	Indicates whether the control is grouping the 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.
LoadXML	Loads an XML document from the specified location, using MSXML parser.
MarkSearchColumn	Retrieves or sets a value that indicates whether the searching column is marked or unmarked
MarkTooltipCells	Retrieves or sets a value that indicates wheter the control marks the cells that have tooltips.
MarkTooltipCellsImage	Specifies a value that indicates the index of icon being displayed in the cells that have tooltips.
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.
ReadOnly	Retrieves or sets a value that indicates whether the control is readonly.
Redo	Redoes the next action in the control's Redo queue.
RedoListAction	Lists the Redo actions that can be performed in the control.
RedoRemoveAction	Removes the first redo actions that can be performed in the control.
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.
SaveXML	Saves the control's content as XML document to the specified location, using the MSXML parser.
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.
SelectByDrag	Specifies whether the user selects multiple items by dragging.
SelectColumnIndex	Retrieves or sets a value that indicates the index of the selected column, if the FullRowSelect property is 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.
ShowFocusRect	Retrieves or sets a value indicating whether the control draws a thin rectangle around the focused item.
ShowImageList	Specifies whether the control's image list window is visible or hidden.

ShowLockedItems	Retrieves or sets a value that indicates whether the locked/fixed items are visible or hidden.
ShowToolTip	Shows the specified tooltip at given position.
SingleSel	Retrieves or sets a value that indicates whether the control supports single or multiple selection.
SingleSort	Returns or sets a value that indicates whether the control supports sorting by single or multiple columns.
SortBarCaption	Specifies the caption being displayed on the control's sort bar when the sort bar contains no columns.
SortBarColumnWidth	Specifies the maximum width a column can be in the control's sort bar.
SortBarHeight	Retrieves or sets a value that indicates the height of the control's sort bar.
SortBarVisible	Retrieves or sets a value that indicates whether control's sort bar is visible or hidden.
SortOnClick	Retrieves or sets a value that indicates whether the control sorts automatically the data when the user click on column's caption.
StartBlockUndoRedo	Starts recording the UI operations as a block of undo/redo operations.
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.
TooltipCellsColor	Retrieves or sets a value that indicates the color used to make the cells that have tooltips.
ToolTipDelay	Specifies the time in ms that passes before the ToolTip appears.
ToolTipFont	Retrieves or sets the tooltip's font.
ToolTipPopDelay	Specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control.
ToolTipWidth	Specifies a value that indicates the width of the tooltip window, in pixels.

ToTemplate	Generates the control's template.
TreeColumnIndex	Retrieves or sets a value that indicates the index of column where the hierarchy lines are displayed.
UnboundHandler	Specifies the control's unbound handler.
Undo	Performs the last Undo operation.
UndoListAction	Lists the Undo actions that can be performed in the control.
UndoRedoQueueLength	Gets or sets the maximum number of Undo/Redo actions that may be stored to the control's queue.
UndoRemoveAction	Removes the last the undo actions that can be performed in the control.
Ungroup	Ungroups the columns, if they have been previously grouped.
UseTabKey	Retrieves or sets a value indicating whether the control uses tab key for changing the searching column.
UseVisualTheme	Specifies whether the control uses the current visual theme to display certain UI parts.
Version	Retrieves the control's version.
ViewMode	Specifies how the data is displayed on the control's view.
ViewModeOption	Specifies options for the control's view mode.
VirtualMode	Specifies a value that indicates whether the control is running in the virtual mode.
VisualAppearance	Retrieves the control's appearance.
VisualDesign	Invokes the control's VisualAppearance designer.
WordFromPoint	Retrieves the word from the cursor.

property Grid.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 Grid.AllowGroupBy as Boolean

Indicates whether the control supports Group-By view.

Type	Description
Boolean	A Boolean expression that specifies whether the user can group the items.

By default, the AllowGroupBy property is False. Set the AllowGroupBy property on True, to allow the user to group the items by dragging the column's header to control's sort bar. The SortBarVisible property specifies whether the control's sort bar is visible or hidden. If the control's sort bar is visible, the user can drag and drop columns to it, so the column get sorted and items grouped. The AddGroupItem event is fired when a new grouping items is added to the control's list. *You can use the AddGroupItem event, to add headers or footers during grouping, customize the aggregate formula to be displayed on different columns, while dropping a column to the sortbar.* The Column.AllowGroupBy property may be used to prevent grouping a specific column. The AllowSort property indicates whether the user can sort a column by clicking the column's header. The IsGrouping property specifies whether the control is grouping/ungrouping items. During grouping, the control keeps the items indentation, in other words, a child item will be a child after or before grouping. The LayoutChanged event is fired when the user changes the layout of the control, including dragging a column to the sort bar. The SortBarColumnsCount property indicates the number of the columns being grouped. The SortBarColumn property indicates the column being sorted giving its position in the sort bar. The Group/Ungroup method groups or ungroup the control's list. For instance, you can remove the grouping items, by calling the Ungroup method. The GroupByTotalField property determines the formula to be applied to the column when it gets grouped. The GroupByFormatCell property determines the format of the cell to be displayed in the grouping item, when the column gets sorted.

The following movies show how Group-By works:

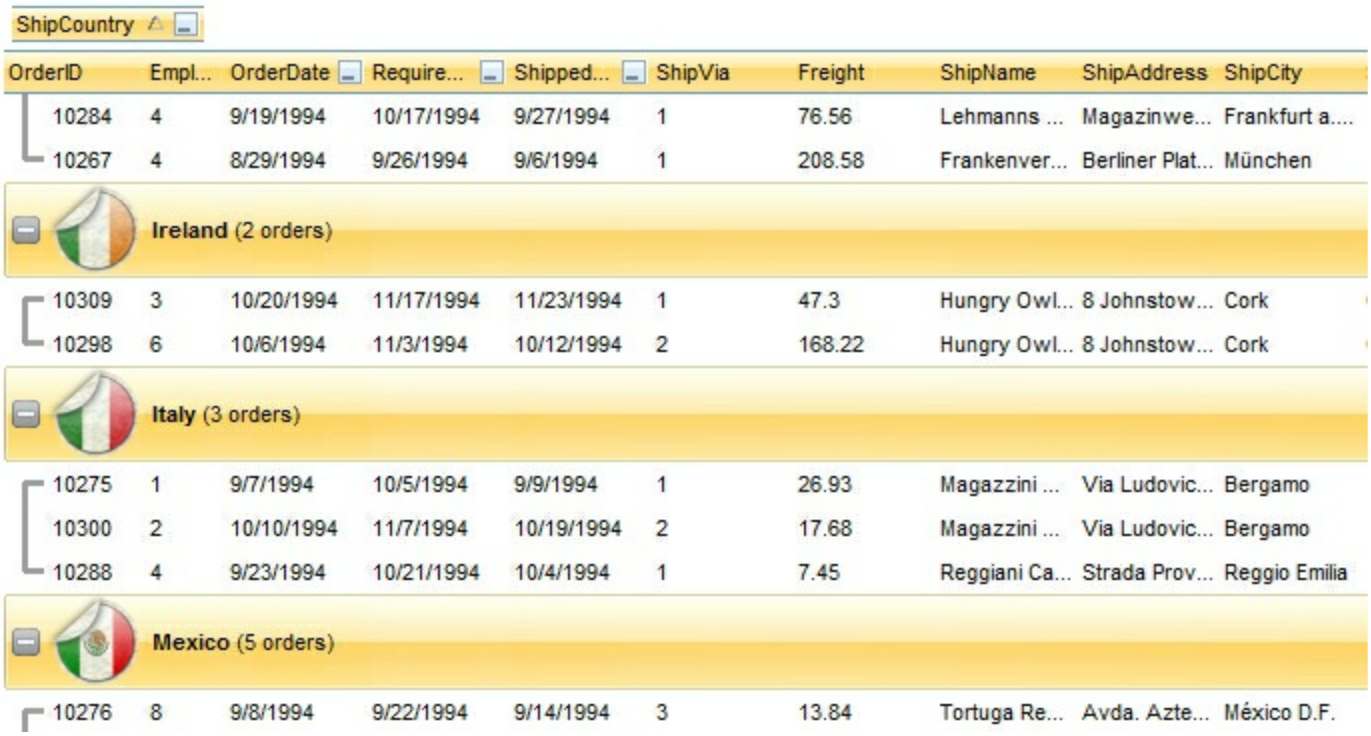
- Group By support - the user can drag and drop one or more columns to the sort bar or group-by bar so the columns get sorted and grouped accordingly.
- Keep Indent - You can keep the indentation of the sub-items/children, once the user groups the rows.
- Header/Footer - Headers and footers support, to display aggregate functions like sum, min, max, and so on.
- CRD support - Can be combined with the exCRD, so you can have the rows being arranged the way you want.

In case you need more than a Group-By feature, you should check the Exontrol's eXPivot.

The Exontrol's eXPivot tool is our approach to provide data summarization, as a pivot table. A pivot-table can automatically sort, count, total or give the average of the data stored in one table or spreadsheet. The user sets up and changes the summary's structure by dragging and dropping fields graphically. The eXPivot component lets the user changes its visual appearance using skins, each one providing an additional visual experience that enhances viewing pleasure. Skins are relatively easy to build and put on any part of the control.

The following screen shot shows the control grouping the orders and details by country (the items keep their children when the control performs the grouping/ungrouping):

The following screen shot shows the control grouping the orders by country:



The AllowGroupBy property/feature has no effect if:

- the control is running in virtual mode (VirtualMode property is True)
- the UnboundHandler property refers to a IUnboundHandler object
- ViewMode property is not exTableView.
- SingleSort property is True.

property Grid.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 Grid.AllowUndoRedo as Boolean

Enables or disables the control's Undo/Redo feature.

Type	Description
Boolean	A Boolean expression that specifies whether the Undo/Redo operations are enabled or disabled.

By default, the AllowUndoRedo property is False. The Undo and Redo features let you remove or repeat single or multiple actions, but all actions must be undone or redone in the order you did or undid them you cant skip actions. For example, if you change the value of three cells in an item and then decide you want to undo the first change you made, you must undo all three changes. To undo an action you need to press Ctrl+Z, while for to redo something you've undone, press Ctrl+Y. Setting the AllowUndoRedo property, clears the previously undo-redo queue. If the AllowUndoRedo property is True, the CTRL+Z performs the last undo operation, while the CTRL+Y redoes the next action in the control's Redo queue. The CanUndo property retrieves a value that indicates whether the control may perform the last Undo operation. The CanRedo property retrieves a value that specifies whether the control can execute the next operation in the control's Redo queue. Call the Undo method to Undo the last control operation. The Redo redoes the next action in the control's redo queue. The UndoRedoQueueLength property gets or sets the maximum number of Undo/Redo actions that may be stored to the control's queue, or in other words how many operations the control's Undo/Redo manager may store.

The records of the Undo/Redo queue may contain actions in the following format:

- "**AddItem**;ITEMINDEX", indicates that a new item has been created
- "**RemoveItem**;ITEMINDEX", indicates that an item has been removed
- "**ChangeItemPos**;ITEMINDEX", indicates that an item changes its position or / and parent
- "**ChangeCellValue**;ITEMINDEX;CELLINDEX", indicates that the cell's value has been changed
- "**ChangeCellState**;ITEMINDEX;CELLINDEX", indicates that the cell's state has been changed

Also, the Undo/Redo queue may include:

- "**StartBlock**", specifies that a block of operations begins
- "**EndBlock**", specifies that a block of operations ends

The URChange(exUndo/exRedo) event notifies your application whenever an Undo/Redo operation is performed. The UndoListAction property lists the Undo actions that can be performed in the control. The RedoListAction property lists the Redo actions that can be performed in the control. Use the UndoRemoveAction method to remove the last actions

from the undo queue. The RedoRemoveAction method removes the first action to be performed if the Redo method is invoked.

property Grid.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 WordFromPoint property determines the word from the cursor.

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

```
Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
  With Grid1
    .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 AxGrid1_MouseMoveEvent(ByVal sender As System.Object, ByVal e As
AxEXGRIDLib._IGridEvents_MouseMoveEvent) Handles AxGrid1.MouseMoveEvent
  With AxGrid1
    .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 axGrid1_MouseMoveEvent(object sender,
AxEXGRIDLib._IGridEvents_MouseMoveEvent e)
{
   axGrid1.ShowToolTip(axGrid1.get_AnchorFromPoint(-1, -1));
}
```

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

```
void OnMouseMoveGrid1(short Button, short Shift, long X, long Y)
{
   COleVariant vtEmpty; V_VT( &vtEmpty ) = VT_ERROR;
   m_grid.ShowToolTip( m_grid.GetAnchorFromPoint( -1, -1 ), vtEmpty, vtEmpty, vtEmpty
);
}
```

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

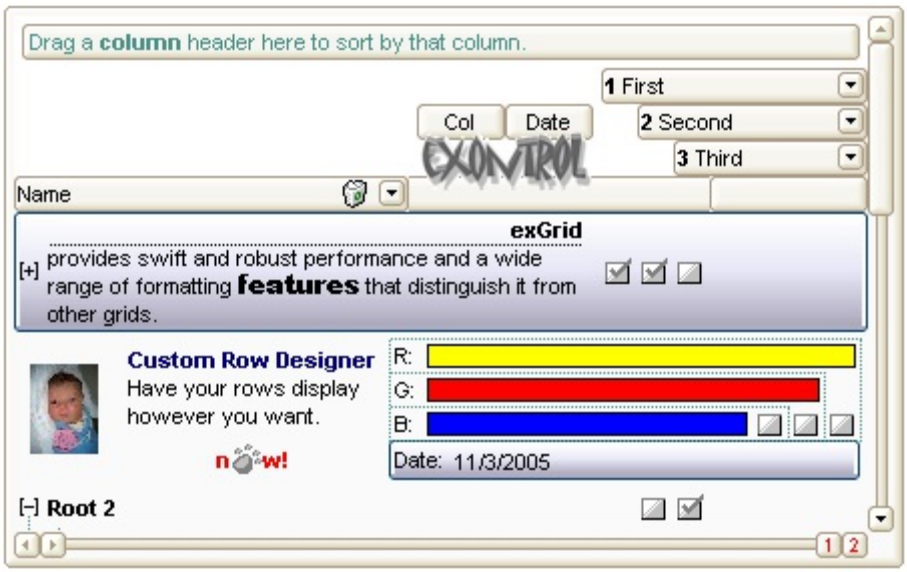
```
*** ActiveX Control Event ***
LPARAMETERS button, shift, x, y

with thisform
   With .Grid1
      .ShowToolTip(.AnchorFromPoint(-1, -1))
   EndWith
endwith
```

property Grid.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 normal.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 Grid1
```

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

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

```
With AxGrid1
   .BeginUpdate()
   .VisualAppearance.Add(&H16, "c:\temp\normal.ebn")
   .Appearance = &H16000000
   .BackColor = Color.FromArgb(250, 250, 250)
   .EndUpdate()
End With
```

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

```
axGrid1.BeginUpdate();
axGrid1.VisualAppearance.Add(0x16, "c:\\temp\\normal.ebn");
axGrid1.Appearance = (EXGRIDLib.AppearanceEnum)0x16000000;
axGrid1.BackColor = Color.FromArgb(250, 250, 250);
axGrid1.EndUpdate();
```

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

```
m_grid.BeginUpdate();
m_grid.GetVisualAppearance().Add( 0x16, COleVariant( "c:\\temp\\normal.ebn" ) );
m_grid.SetAppearance( 0x16000000 );
m_grid.SetBackColor( RGB(250,250,250) );
m_grid.EndUpdate();
```

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

```
with thisform.Grid1
   .BeginUpdate
      .VisualAppearance.Add(0x16, "c:\temp\normal.ebn")
      .Appearance = 0x16000000
```

```
        .BackColor = RGB(250, 250, 250)
    .EndUpdate
endwith
```

method Grid.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 FilterBarDropDownHeight property to specify the height of the drop down filter window. Use the FilterInclude property to specify whether the child items should be included to the list when the user applies the filter. Use the FilterCriteria property to filter items using the AND, OR and NOT operators. Use the ShowFilter method to show programmatically the column's drop down filter window.

property Grid.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. If you want to make the auto search feature case sensitive you have to use ASCIIUpper = "" . By default, the ASCIILower property is = "abcdefghijklmnopqrstuvwxyz?שבהפתס?עצפלמנטכרחאוהגי"

property Grid.ASCIIUpper as String

Specifies the set of upper characters.

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

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

method Grid.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 Grid1_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.

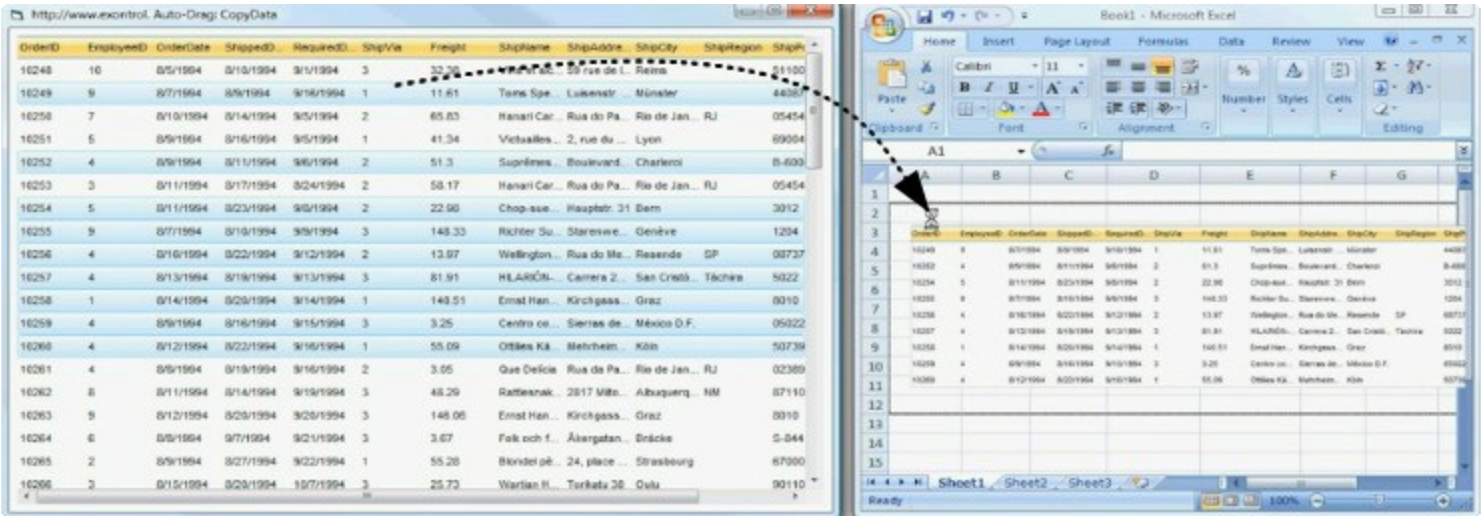
The advantage of the AttachTemplate relative to Template / ExecuteTemplate is that the AttachTemplate can add handlers to the control events.

property Grid.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 Grid.AutoEdit as Boolean

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

Type	Description
Boolean	A boolean expression that indicates whether the editing operation starts once that a cell is focused.

Use the AutoEdit property to choose how the user edits the data. By default, the AutoEdit property is True. Use the Edit method to start editing the focused cell. Use the EditType property to define the column's editor. Use the ReadOnly property to make the control read only. Use the FocusItem property to retrieve the focused item. Use the FocusColumnIndex property to get the index of the column that's focused. Use the Editing property to check whether the control is in edit mode, or to get the window's handle for the built-in editor that's visible and focused.

The edit events are fired in the following order:

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

The following VB sample starts editing a cell when user presses the F4 key:

```
Private Sub Grid1_KeyDown(KeyCode As Integer, Shift As Integer)
  ' Starts editing data when the user presses the F4 key
  With Grid1
    If (Not .AutoEdit) Then
      If (KeyCode = vbKeyF4) Then .Edit
    End If
  End With
End Sub
```

The following C++ sample starts editing the focused cell when user presses the F4 key:

```
void OnKeyDownGrid1(short FAR* KeyCode, short Shift)
```

```
{
   if ( *KeyCode == VK_F4 )
   {
      COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
      if ( m_grid.GetEditing() == 0 )
         m_grid.Edit( vtMissing );
   }
}
```

The following VB.NET sample starts editing the focused cell when user presses the F4 key:

```
Private Sub AxGrid1_KeyDownEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_KeyDownEvent) Handles AxGrid1.KeyDownEvent
   If (Convert.ToUInt32(e.keyCode) = Convert.ToUInt32(Keys.F4)) Then
      AxGrid1.Edit(Nothing)
   End If
End Sub
```

The following C# sample starts editing the focused cell when user presses the F4 key:

```
private void axGrid1_KeyDownEvent(object sender,
AxEXGRIDLib._IGridEvents_KeyDownEvent e)
{
   if (Convert.ToUInt32(e.keyCode) == Convert.ToUInt32(Keys.F4))
      axGrid1.Edit(null);
}
```

The following VFP sample starts editing the focused cell when user presses the F4 key:

```
private void axGrid1_KeyDownEvent(object sender,
AxEXGRIDLib._IGridEvents_KeyDownEvent e)
{
   if (Convert.ToUInt32(e.keyCode) == Convert.ToUInt32(Keys.F4))
      axGrid1.Edit(null);
}
```

property Grid.AutoSearch as Boolean

Enables or disables the incremental searching feature.

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

By default, the AutoSearch property is True. The auto-search feature is is commonly known as incremental search. An incremental search begins searching as soon as you type the first character of the search string. As you type in the search string, the control selects the item (and highlight the portion of the string that match where the string (as you have typed it so far) would be found. The control supports 'starts with' or 'contains' incremental search as described in the AutoSearch property of the Column object. Use the ASCIILower and ASCIIUpper properties to specify the set of lower and upper characters when auto search feature is enabled. Use the ExpandOnSearch property to expand items automatically while user types characters to search for a specific item. Use the MarkSearchColumn property to specify whether the control draws a rectangle around the searching column. The SearchColumnIndex property determines the index of the searching column.

The control highlights the characters as the user types them:

property Grid.BackColor as Color

Retrieves or sets a value that indicates the control's background color.

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

Use the BackColor property to set the control's background color. If the control contains locked columns, (if the CountLockedColumns property is grater than 0, a locked column is a column non scrolable), use the BackColorLock property to specify the background color for locked columns. Use the CellBackColor property to set the cell's background color. Use the ItemBackColor property to specify the item's background color. The control highlights the selected items only if the SelBackColor and BackColor properties have different values, and the SelForeColor and ForeColor properties have different values. Use the Def(exCellBackColor) property to specify the background color for all cells in the column.

The following VB sample sets the background color for the first column:

```
With Grid1.Columns(0)
   .Def(exCellBackColor) = RGB(240, 240, 120)
End With
```

The following C++ sample sets the background color for the first column:

```
#include "Column.h"
#include "Columns.h"
CColumns columns = m_grid.GetColumns();
CColumn column = columns.GetItem( COleVariant( long(0) ) );
column.SetDef(4, COleVariant( (long)RGB(240,240,240 ) ) );
```

The following VB.NET sample sets the background color for the first column:

```
With AxGrid1.Columns(0)
   .Def(EXGRIDLib.DefColumnEnum.exCellBackColor) = ToUInt32(Color.FromArgb(240,
240, 240))
End With
```

where the ToUInt32 function converts a Color expression to 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 sets the background color for the first column:

```
axGrid1.Columns[0].set_Def(EXGRIDLib.DefColumnEnum.exCellBackColor,
ToUInt32(Color.FromArgb(240, 240, 240)));
```

where the ToUInt32 function converts a Color expression to 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 VFP sample sets the background color for the first column:

```
with thisform.Grid1.Columns(0)
    .Def(4) = RGB(240,240,240)
endwith
```

property Grid.BackColorAlternate as Color

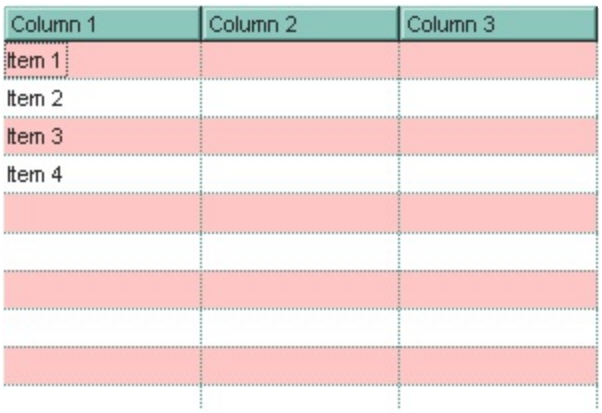
Specifies the background color used to display alternate items in the control.

Type	Description
Color	A color expression that indicates the alternate background color. If the first byte of four is 7F, the color is applied to the items section only. For instance, a value of 0x7F0000FF indicates that the BackColorAlternate property is red, and it applied to the items section only, so the non-items section is not painted.

By default, the control's BackColorAlternate property is zero. Use the BackColorAlternate property to specify the background color used to display alternate items in the control. The control ignores the BackColorAlternate property if it is 0 (zero). Use the BackColor property to specify the control's background color. Use the SelBackColor property to specify the selection background color. Use the ItemBackColor property to specify the item's background color. Use the CellBackColor property to specify the cell's background color. Use the Def(exCellBackColor) property to specify the background color for all cells in the column. If the first two bytes of the BackColorAlternate property are 0x7F, the non-items area is not filled.

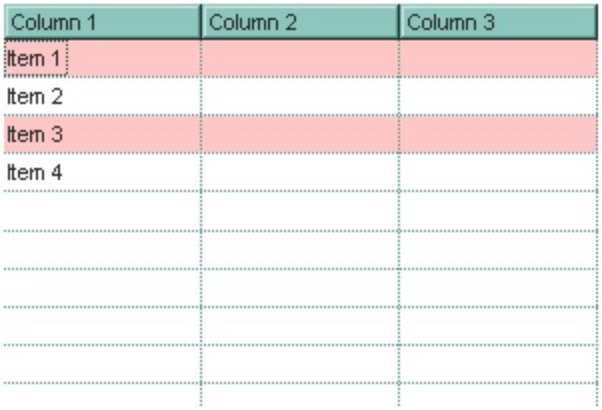
For instance, the following VB sample draws alternate rows, including the non-items area:

```
Grid1.BackColorAlternate = RGB(255, 190, 190)
```



The following VB sample draws alternate rows, excluding the non-items area:

```
Grid1.BackColorAlternate = &H7F000000 Or RGB(255, 190, 190)
```



The following VB.NET sample draws alternate rows, excluding the non-items area:

```
Dim sRGB As UInt32 = &H7F000000 Or ToUInt32(ControlPaint.LightLight(Color.Lavender))
AxGrid1.Template = "BackColorAlternate = " + sRGB.ToString()
```

or use the BackColorAlternate32 property provided by the /NET assembly version

where the ToUInt32 method converts a Color expression to a unsigned integer:

```
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 draws alternate rows, excluding the non-items area:

```
UInt32 sRGB = 0x7F000000 | ToUInt32( ControlPaint .LightLight(Color.Lavender) );
axGrid1.Template = "BackColorAlternate = " + sRGB.ToString();
```

where the ToUInt32 method converts a Color expression to a unsigned integer.

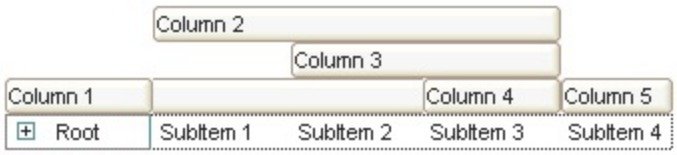
```
private UInt32 ToUInt32(Color c)
{
   long i;
   i = c.R;
   i = i + 256 * c.G;
   i = i + 256 * 256 * c.B;
   return Convert.ToUInt32(i);
```

property Grid.BackColorHeader as Color

Specifies the header's background color.

Type	Description
Color	A color expression that indicates the background color of the control's header bar. The last 7 bits in the high significant byte of the color indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

Use the BackColorHeader and ForeColorHeader properties to define colors used to paint the control's header bar. Use the HeaderVisible property to show or hide the control's header. Use the HeaderHeight property to specify the height of the control's header bar. Use the LevelKey property to allow multiple levels header bar. 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 header 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 Grid1
   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_grid.GetVisualAppearance().Add( 0x24,
COleVariant(_T("D:\\Temp\\ExGrid.Help\\header.ebn")) );
m_grid.SetBackColorHeader( 0x24000000 );
```

The following VB.NET sample changes the visual aspect of the control' header bar:

```
With AxGrid1
   With .VisualAppearance
      .Add(&H24, "D:\Temp\ExGrid.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:

```
axGrid1.VisualAppearance.Add(0x24, "D:\\Temp\\ExGrid.Help\\header.ebn");
axGrid1.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.Grid1
   With .VisualAppearance
      .Add(36, "D:\Temp\ExGrid.Help\header.ebn")
   EndWith
   .BackColorHeader = 603979776
EndWith
```

property Grid.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 Grid.BackColorLock as Color

Retrieves or sets a value that indicates the control's background color for the locked area.

Type	Description
Color	A color expression that indicates the background color for locked columns.

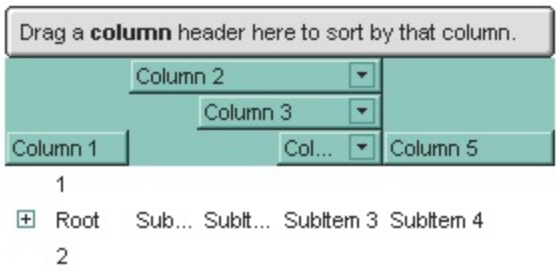
The control contains locked columns if the CountLockedColumn property is grater than zero (0). A locked column is fixed to the left side of the control, and it cannot be scrolled. If the CountLockedColumn property is greater than 0, the BackColor property sets the background color for the unlocked area. The unlocked area is the area that contains scrollable columns. Use the Def(exCellBackColor) property to specify the background color for all cells in the column. Use the CellBackColor property to set the cell's background color.

property Grid.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.

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 Grid1
   .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_grid.GetVisualAppearance().Add( 0x60,
COleVariant(_T("D:\\Temp\\ExGrid.Help\\sortbar.ebn")) );
m_grid.SetSortBarVisible( TRUE );
m_grid.SetBackColorSortBar( 0x60000000 );
m_grid.SetBackColorSortBarCaption( m_grid.GetBackColorSortBar() );
```

The following VB.NET sample changes the appearance for the control's sort bar:

```
With AxGrid1
   .SortBarVisible = True
   With .VisualAppearance
      .Add(&H60, "D:\Temp\ExGrid.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:

```
axGrid1.VisualAppearance.Add(0x60, "D:\\Temp\\ExGrid.Help\\sortbar.ebn");
axGrid1.SortBarVisible = true;
axGrid1.Template = "BackColorSortBar = 1610612736";
axGrid1.Template = "BackColorSortBarCaption = 1610612736";
axGrid1.ForeColorSortBar = Color.Black;
```

The following VFP sample changes the appearance for the control's sort bar

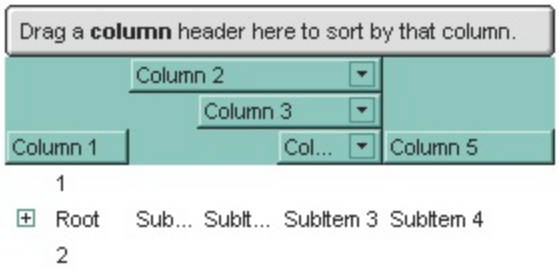
```
With thisform.Grid1
   With .VisualAppearance
      .Add(96, "D:\Temp\ExGrid.Help\sortbar.ebn")
   EndWith
   .SortBarVisible =.t.
   .BackColorSortBar = 1610612736
   .BackColorSortBarCaption = .BackColorSortBar
   .ForeColorSortBar = 0
EndWith
```

property Grid.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.

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 Grid1
   .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_grid.GetVisualAppearance().Add( 0x60,
COleVariant(_T("D:\\Temp\\ExGrid.Help\\sortbar.ebn")) );
m_grid.SetSortBarVisible( TRUE );
m_grid.SetBackColorSortBar( 0x60000000 );
```

```
m_grid.SetBackColorSortBarCaption( m_grid.GetBackColorSortBar() );
```

The following VB.NET sample changes the appearance for the control's sort bar:

```
With AxGrid1
   .SortBarVisible = True
   With .VisualAppearance
      .Add(&H60, "D:\Temp\ExGrid.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:

```
axGrid1.VisualAppearance.Add(0x60, "D:\\Temp\\ExGrid.Help\\sortbar.ebn");
axGrid1.SortBarVisible = true;
axGrid1.Template = "BackColorSortBar = 1610612736";
axGrid1.Template = "BackColorSortBarCaption = 1610612736";
axGrid1.ForeColorSortBar = Color.Black;
```

The following VFP sample changes the appearance for the control's sort bar

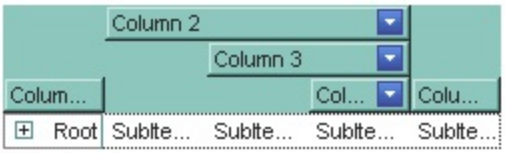
```
With thisform.Grid1
   With .VisualAppearance
      .Add(96, "D:\Temp\ExGrid.Help\sortbar.ebn")
   EndWith
   .SortBarVisible =.t.
   .BackColorSortBar = 1610612736
   .BackColorSortBarCaption = .BackColorSortBar
   .ForeColorSortBar = 0
EndWith
```

property Grid.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 Grid1
   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_grid.GetVisualAppearance().Add( 0x01,
COleVariant(_T("D:\\Temp\\ExGrid.Help\\fbardd.ebn")) );
```

```
m_grid.SetBackground( 0 /*exHeaderFilterBarButton*/, 0x1000000 );
```

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

```
With AxGrid1
   With .VisualAppearance
      .Add(&H1, "D:\Temp\ExGrid.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:

```
axGrid1.VisualAppearance.Add(0x1, "D:\\Temp\\ExGrid.Help\\fbardd.ebn");
axGrid1.set_Background(EXTREELib.BackgroundPartEnum.exHeaderFilterBarButton,
0x1000000);
```

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

```
With thisform.Grid1
   With .VisualAppearance
      .Add(1, "D:\Temp\ExGrid.Help\fbardd.ebn")
   EndWith
   .Object.Background(0) = 16777216
EndWith
```

The 16777216 value is the 0x1000000 value in hexadecimal

method Grid.BeginUpdate ()

Maintains performance when items are added to the control one at a time.

Type	Description

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

The following VB sample prevents the control from painting while it adds one column, and one item, using the BeginUpdate and EndUpdate methods:

```
With Grid1
   .BeginUpdate
      .LinesAtRoot = LinesAtRootEnum.exLinesAtRoot
      .FullRowSelect = False
      .DefaultItemHeight = 24
      With .Columns.Add("Mask")
         With .Editor
            .EditType = EditTypeEnum.MaskType
            .Appearance = SingleApp
            .Mask = "{0,255}\.{0,255}\.{0,255}\.{0,255}"
         End With
      End With
      .Items.AddItem "193.226.40.161"
   .EndUpdate
End With
```

The following C++ sample prevents the control from painting while adding columns and items:

```
#include "Column.h"
#include "Columns.h"
m_grid.BeginUpdate();
m_grid.SetColumnAutoResize( FALSE );
CColumns columns = m_grid.GetColumns();
for ( long i = 0; i < 4; i++ )
   columns.Add( "NewColumn" );
```

```
CItems items = m_grid.GetItems();
for ( long j = 0; j < 10; j++ )
{
   COleVariant vtItem( items.AddItem( COleVariant( "new item " ) ) );
   for ( long k = 1 ; k < 4; k++ )
      items.SetCellValue( vtItem, COleVariant( k ), COleVariant( "new item " ) );
}
m_grid.EndUpdate();
```

The following VB.NET sample prevents the control from painting while adding columns and items:

```
With AxGrid1
   .BeginUpdate()
   .LinesAtRoot = EXGRIDLib.LinesAtRootEnum.exLinesAtRoot
   .FullRowSelect = False
   .DefaultItemHeight = 24
   With .Columns.Add("Mask")
      With .Editor
         .EditType = EXGRIDLib.EditTypeEnum.MaskType
         .Appearance = EXGRIDLib.AppearanceEnum.Etched
         .Mask = "{0,255}\.{0,255}\.{0,255}\.{0,255}"
      End With
   End With
   .Items.AddItem("193.226.40.161")
   .EndUpdate()
End With
```

The following C# sample prevents the control from painting while adding columns and items:

```
axGrid1.BeginUpdate();
EXGRIDLib.Columns columns = axGrid1.Columns;
columns.Add("Column 1");
columns.Add("Column 3");
EXGRIDLib.Items items = axGrid1.Items;
items.AddItem("new item");
items.AddItem("new item");
axGrid1.EndUpdate();
```

The following VFP sample prevents the control from painting while adding columns and items:

```
with thisform.Grid1
   .BeginUpdate
      .LinesAtRoot = .t.
      .FullRowSelect = .f.
      .DefaultItemHeight = 24
      With .Columns.Add("Mask")
         With .Editor
            .EditType = 8
            .Appearance = 1
            .Mask = "{0,255}\.{0,255}\.{0,255}\.{0,255}"
         EndWith
      EndWith
      .Items.AddItem("193.226.40.161")
   .EndUpdate
endwith
```

property Grid.CanRedo as Boolean

Retrieves a value that indicates whether the control can perform a Redo operation.

Type	Description
Boolean	A Boolean expression that specifies whether the control can perform the next action in the control's Redo queue.

For instance, you can use the CanRedo property to update the Redo button in your toolbar, so the user knows that Redo operations in the control may be performed. The Redo redoes the next action in the control's redo queue. If the AllowUndoRedo property is True, the CTRL+Y redoes the next action in the control's Redo queue. The RedoListAction property lists the Redo actions that can be performed in the control.

property Grid.CanUndo as Boolean

Retrieves a value that indicates whether the control can perform an Undo operation.

Type	Description
Boolean	A Boolean expression that specifies whether the control can perform the last Undo operation.

For instance, you can use the CanUndo property to update the Undo button in your toolbar, so the user knows that Undo operations in the control may be performed. Call the Undo method to Undo the last control operation. By default, the if the AllowUndoRedo property is True, the CTRL+Z performs the last Undo operation. The CanRedo property retrieves a value that specifies whether the control can execute the next operation in the control's Redo queue. The Redo redoes the next action in the control's redo queue. If the AllowUndoRedo property is True, the CTRL+Y redoes the next action in the control's Redo queue. The UndoListAction property lists the Undo actions that can be performed in the control. The RedoListAction property lists the Redo actions that can be performed in the control.

property Grid.CauseValidateValue as ValidateValueType

Returns or sets a value that determines whether the ValidateValue event occurs before the user changes the cell's value.

Type	Description
ValidateValueType	A ValidateValueType expression that indicates whether the ValidateValue event is fired when user leaves the focused cell or focused item.

By default, the CauseValidateValue property is exNoValidate (False). The ValidateValue event is fired only if the CauseValidateValue property is exValidateCell or exValidateItem, once the user alters the focused cell and the user is trying to leave the focused cell or item. Use the exValidateItem option to validate the item once a new item is focused, or use the exValidateCell to validate the cell's value once the user leaves the focused cell. You can use the ValidateValue event to prevent the user enters wrong values to the cells/items. In conclusion, the user can focus a new cell (in the same item) while using validation, only if the CauseValidateValue property is exValidateItem. Else, the user can not move the focus to a new cell or items until the user validates the value (Cancel parameter of the ValidateValue event is False). Call the DiscardValidateValue method to restore back the values being changed during the validation.

The following VB sample displays a message box with Yes, No and Cancel buttons to validate the changed value:

```
Private Sub Grid1_ValidateValue(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As
Long, ByVal NewValue As Variant, Cancel As Boolean)
  Cancel = True
  Dim iResult As Long
  i = MsgBox("Validate GRID1 :" & Grid1.Items.CellCaption(Item, ColIndex ) & " " &
NewValue, vbYesNoCancel)
  If (i = vbCancel) Then
    Grid1.DiscardValidateValue
  Else
    Cancel = i = vbNo
  End If
End Sub
```

The ValidateValue event provides the Cancel parameter which can be used to validate the value being changed. The sample calls the DiscardValidateValue method if user selects Cancel, so the value are being restored. The validation is accepted once the user selects

the Yes button. If No, the validation continues, and if the control's AutoEdit property is True, the control re-opens the editor for validation.

During the validation you may have the following order of the events:

- Edit - prevent showing the editor for specified cell.
- EditOpen - indicates that the editor for the focused cell is being opened.
- EditClose - indicates that the editor for the focused cell is being closed.
- ValidateValue - notifies your application that the value must be validated (Cancel parameter on False)
- Change - notifies the application once the user validates the newly value. In case the control is bounded to a database, the change is performed to the database too.
- Error - notifies the application for any error (for instance, if the change is not supported by the database, the Error indicates the error being issued).

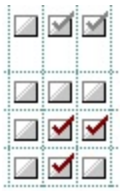
The ValidateValue event is not fired if the CellValue property is called during the event.

property Grid.CheckImage(State as CheckStateEnum) as Long

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

Type	Description
State as CheckStateEnum	A CheckStateEnum expression that defines the state of the check box being changed. 0 - unchecked, 1 - checked, 2 - partial checked.
Long	A long expression that indicates the index of the icon used. 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. If the index is not valid the default icon is used.

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. Use the Images method to load icons at runtime.



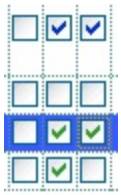
The following VB sample defines icons for the cells of check box type:

```
With Grid1
  .BeginUpdate
    .Columns.Add "Radio"
    For i = 0 To 2
      Dim h As HITEM
      h = .Items.AddItem("Option " & i)
      .Items.CellHasCheckBox(h) = True
      .Items.CellState(h) = i Mod 2
```

```
        Next
        .CheckImage(0) = 1
        .CheckImage(1) = 3
    .EndUpdate
End With
```

The following C++ sample changes the default appearance for check boxes:

```
CString s =
"gBJJgBAIDAAEg4ACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjMLjABAAgjUYkUnlUnAktg8u

s = s +
"Or0esz2u0lxqGIjGc22W2G51u71+jueR4Fl2gAdXFzWZ2/I3G80W952C01x5PH6m65/WwW

s = s +
"EWQJFcoSfAkXPHGMrJ0ucanGAEtRxIMIPpAL9skpUsxtIUjyLDU0w5IrsSrEcZoxLr5yi2rLTtJyM

s = s +
"MxW6dwWOyNk2XZRr2ZUtx2zctDWrHz4znWV523TdqW/PteRJfl90HaVzW87N1wTdt4XeA

s = s + "7C0YAICA";
m_grid.Images(COleVariant(s));
m_grid.SetCheckImage(0,1);
m_grid.SetCheckImage(1,2);
m_grid.SetCheckImage(2,3);
```



The following C# sample changes the default appearance for check boxes:

```
string s =
"gBJJgBAIDAAEg4ACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjMLjABAAgjUYkUnlUnAktg8u

s = s +
"Or0esz2u0lxqGIjGc22W2G51u71+jueR4Fl2gAdXFzWZ2/I3G80W952C01x5PH6m65/WwW
```

```
s = s +
"EWQJFcoSfAkXPHGMrJ0ucanGAEtRxIMIPpAL9skpUsxtIUjyLDU0w5IrsSrEcZoxLr5yi2rLTtJyM

s = s +
"MxW6dwWOyNk2XZRr2ZUtx2zctDWrHz4znWV523TdqW/PteRJfl90HaVzW87N1wTdt4XeA

s = s + "7C0YAICA";
axGrid1.Images(s);
axGrid1.set_CheckImage( 0,1 );
axGrid1.set_CheckImage( 1,2 );
axGrid1.set_CheckImage( 2,3 );
```

The following VB.NET sample changes the default appearance for check boxes:

```
With AxGrid1
   Dim s As String =
"gBJJgBAIDAAEg4ACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjMLjABAAgjUYkUnlUnAktg8u

   s = s +
"Or0esz2u0lxqGIjGc22W2G51u71+jueR4Fl2gAdXFzWZ2/I3G80W952C01x5PH6m65/WwW

   s = s +
"EWQJFcoSfAkXPHGMrJ0ucanGAEtRxIMIPpAL9skpUsxtIUjyLDU0w5IrsSrEcZoxLr5yi2rLTtJyM

   s = s +
"MxW6dwWOyNk2XZRr2ZUtx2zctDWrHz4znWV523TdqW/PteRJfl90HaVzW87N1wTdt4XeA

   s = s + "7C0YAICA"
   .Images(s)
   .set_CheckImage(0, 1)
   .set_CheckImage(1, 2)
   .set_CheckImage(2, 3)
End With
```

The following VFP sample changes the default appearance for check boxes:

```
with thisform.Grid1
   local s
```

```
   s =
"gBJJgBAIDAAEg4ACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjMLjABAAgjUYkUnlUnAktg8u

   s = s +
"rptJq9Lp1ZmdVr1fsEYrthslls1ntEqmaZM0YtlutoAt9yuNzu11vFwvV0sVOu97v98wWBwl5qN

   s = s +
"Or0esz2u0lxqGIjGc22W2G51u71+jueR4Fl2gAdXFzWZ2/I3G80W952C01x5PH6m65/WwW

   s = s +
"tS6hmPw/TwPe/jywSw7/Iw+cHQA7z2wM574KSucHvSzj7PxDYAQ690KQQlkGPU9L5s5AaM

   s = s +
"EWQJFcoSfAkXPHGMrJ0ucanGAEtRxIMIPpAL9skpUsxtIUjyLDU0w5IrsSrEcZoxLr5yi2rLTtJyM

   s = s +
"XSE8SIFUpwPP0/OGbtSS5TEtzpQkb0O71FKHM1UVPL1Cwyy0iPwzjQ0qjNL1jX1Z1XTlPTu1

   s = s +
"MxW6dwWOyNk2XZRr2ZUtx2zctDWrHz4znWV523TdqW/PteRJfl90HaVzW87N1wTdt4XeA

   s = s +
"meWZqw0f5pI2bZhayEZStCuK3oSsaGrWiaPo2k6DpGI6Up2frOiofJXqaOo+kKMpIkiNEAtLX

   s = s + "7C0YAICA"
   .Images(s)
   .CheckImage(0) = 1
   .CheckImage(1) = 2
   .CheckImage(2) = 3
endwith
```

The following template can be placed on the control's Template page:

```
Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl

CheckImage(1) = 2
CheckImage(0) = 1
```

```
CheckImage(2) = 3
```

method Grid.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 ShowFilter method to show programmatically the column's drop down filter window.

property Grid.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.

By default, the ColumnAutoResize property is True. Use the ColumnAutoResize property to fit all visible columns in the control's client area. Use the Add method to add new columns to the control's Columns collection. Use the Width property to change the column's width. Use the Visible property to hide a column. Use the ContinueColumnScroll property to specify whether the user scrolls the control's content column by column or pixel by pixel. If the ColumnAutoResize property is True, the control does not display the control's horizontal scroll bar. Use the ScrollBars property to show or hide the control's scroll bars. By default, the control adds scroll bars when required.

property Grid.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 get the item or cell from the cursor. Use the HeaderVisible property to show or hide the control's header. The ColumnFromPoint property returns -1 if there is no column's caption at the cursor position. Use the Visible property to hide a particular column. Use the LevelKey property to allow multiple levels header bar. Use the HeaderHeight property to specify the height of the control's header bar. Use the BackColorHeader property to specify the header's background color. Use the AllowSizing property to disable resizing a column when user clicks the right margin of the column. Use the SortOnClick property to specify the action that control takes when the column's caption is clicked. The Background(exCursorHoverColumn) property specifies the visual appearance of the column's header when the cursor hovers it. The WordFromPoint property determines the word from the cursor.

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

```
Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  With Grid1
```

```
    Dim c As Long
    c = .ColumnFromPoint(-1, -1)
    If (c >= 0) Then
      With .Columns(c)
        Debug.Print .Caption
      End With
    End If
  End With
End Sub
```

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

```
Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
  With Grid1
    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 OnMouseMoveGrid1(short Button, short Shift, long X, long Y)
{
  long nColIndex = m_grid.GetColumnFromPoint( X, Y );
  if ( nColIndex >= 0 )
  {
    CColumn column = m_grid.GetColumns().GetItem( COleVariant( nColIndex ) );
    OutputDebugString( column.GetCaption() );
  }
}
```

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

```
Private Sub AxGrid1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_MouseMoveEvent) Handles AxGrid1.MouseMoveEvent
   With AxGrid1
      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 axGrid1_MouseMoveEvent(object sender,
AxEXGRIDLib._IGridEvents_MouseMoveEvent e)
{
   int i = axGrid1.get_ColumnFromPoint( e.x,e.y );
   if ( i >= 0 )
      System.Diagnostics.Debug.WriteLine( axGrid1.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.Grid1
   i = .ColumnFromPoint( x, y )
   if ( i >= 0 )
      wait window nowait .Columns(i).Caption
   endif
endwith
```

property Grid.Columns as Columns

Retrieves the control's column collection.

Type	Description
Columns	The control's Columns object.

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.

The following VB sample adds two new columns to the control:

```
With Grid1
   .Columns.Add "Column 1"
   With .Columns.Add("Column 2")
      With .Editor
         .EditType = CalculatorType
      End With
   End With
End With
```

The following C++ sample adds two new columns to the control:

```
#include "Column.h"
#include "Columns.h"
COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
CColumns columns = m_grid.GetColumns();
columns.Add( "Column 1" );
CColumn column( V_DISPATCH( &columns.Add( "Column 2" ) ) );
CEditor editor = column.GetEditor();
editor.SetEditType( 21 /*CalculatorType*/ );
```

The following VB.NET sample adds two new columns to the control:

```
With AxGrid1
   .Columns.Add("Column 1")
```

```
    Dim c As EXGRIDLib.Column = .Columns.Add("Column 2")
    With c.Editor
        .EditType = EXGRIDLib.EditTypeEnum.CalculatorType
    End With
End With
```

The following C# sample adds two new columns to the control:

```
EXGRIDLib.Column column = axGrid1.Columns.Add("Column 1") as EXGRIDLib.Column ;
column = axGrid1.Columns.Add("Column 2") as EXGRIDLib.Column;
column.Editor.EditType = EXGRIDLib.EditTypeEnum.CalculatorType;
```

The following VFP sample adds two new columns to the control:

```
with thisform.Grid1
    .Columns.Add("Column 1")
    With .Columns.Add("Column 2")
        with .Editor
            .EditType = 21 && CalculatorType
        endwith
    EndWith
endwith
```

property Grid.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 Grid.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 Grid.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 control's Columns float-bar is visible or hidden.

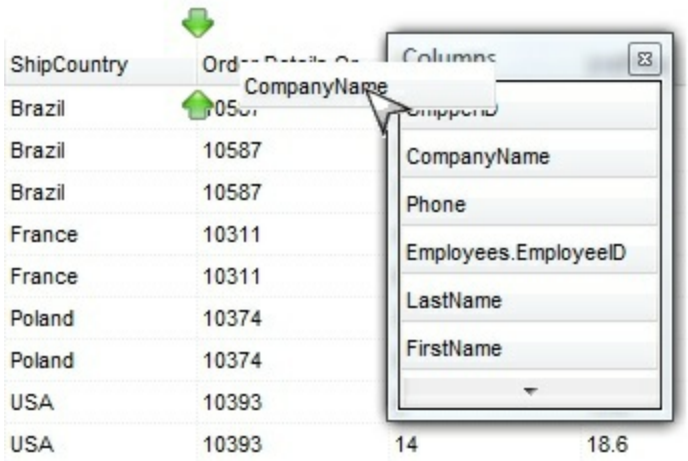
The ColumnsFloatBarVisible property indicates whether the control displays a floating panel that shows the hidden columns, so the user can drag and drop columns on order to show or hide the columns from the control. Use the ColumnsFloatBarSortOrder property to sort the columns to be displayed in the 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 Grid.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 CellValue 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 Grid.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.

Use the ContinueColumnScroll property to define how the control scrolls the columns. Use the EnsureVisibleColumn method scrolls the control's content to ensure that the column fits the client area. Use the Scroll method to scroll the control's columns, column by column, if the ContinueColumnScroll property is False. 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.

method Grid.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. Use the CopyTo method to copy the control's view to an EMF file. 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. You can use the Export method to export the control's DATA in CSV format.

The following VB sample saves the control's content to a EMF file, when user presses the CTRL+C key:

```
Private Sub Grid1_KeyDown(KeyCode As Integer, Shift As Integer)
   If (KeyCode = vbKeyC) And Shift = 2 Then
      Clipboard.Clear
      Grid1.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 AxGrid1
  .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;
axGrid1.Copy();
```

property Grid.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. You can use the Export method to export the control's DATA in CSV format. Use the Copy method to copy the control's content to the clipboard.

- 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 (Grid1.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 Grid1.CopyTo("")
   Debug.Print i
Next
```

property Grid.CountLockedColumns as Long

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

Type	Description
Long	A long expression that indicates the number of locked columns.

The control is able to display two types of columns: locked and unlocked columns. A locked column is not scrollable, and it is fixed to the left side of the control. An unlocked control is scrollable. Use the CountLockedColumns property to define the number of columns that are in the locked area. Use the BackColorLock property to specify the background color for the locked area. Use the ForeColorLock property to specify the foreground color for the locked area. If the CountLockedColumns property is zero, no locked columns defined, then BackColorLock and ForeColorLock have no effect. Use the LockedItemCount property to lock or unlock items to the top or bottom side of the control. Use the MergeCells method to combine one or more cells in a single cell.

property Grid.DataSource as Object

Retrieves or sets a value that indicates the data source for the object.

Type	Description
Object	An Object that defines the control's data. Currently, the control accepts ADO.Recordset, ADODB.Recordset objects, DAO recordsets

The DataSource property binds the control to an ADO, ADODB or DAO recordset. Setting the DataSource property clears the control's columns collection. The DataSource property adds a column for each field find in the recordset. Depending on type of the field, the control sets the column's EditType property. For instance, an field of "OLE Object" type is converted to PictureType edit type, a field of Date type, uses DateType edit type. The DetectAddNew property detects adding new records to a recordset. The DetectDelete property detects removing records from the recordset.

The DataSource property can load all data in the memory or just visible records (virtual mode). The VirtualMode property indicates whether the control loads all records in memory or just visible records. If the VirtualMode property is False (by default), all records are loaded in memory. The user must call the VirtualMode property before setting the DataSource, else an error occurs. If the VirtualMode property is True, before specifying the DataSource, the control loads virtually the records, just visible records are loaded. Use the control's virtual mode when you require to display and edit large databases, and you don't want to load the entire database in memory. Aldo, running the virtual mode disables some features including sorting and filtering, like explained in the VirtualMode property. The control builds an internal object that implements the IUnboundHandler interface that provides data for the control, when running in virtual mode, so the UnboundHandler property is not empty.

The following template script loads virtually the Order table, using the Template feature of the control (copy the following template and paste it to the control's WYSWYG Template editor) (the sample uses Jet.OLEDB provider to handle MDB files) :

```
Dim rs
VirtualMode = True
ColumnAutoResize = False
rs = CreateObject("ADOR.Recordset")
{
   Open("Orders","Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program
Files\Exontrol\ExGrid\Sample\SAMPLE.MDB", 3, 3 )
}
```

```
DataSource = rs
ConditionalFormats
{
   Add("%1 > 4").Bold = True
   Add("%1 = 1 or %1 = 3")
   {
      Underline = True
      ForeColor = RGB(255,0,0)
      ApplyTo = 1
   }
}
```

or (the sample uses VFPOLEDB provider to handle DBF files)

```
Dim rs
VirtualMode = True
ColumnAutoResize = False
rs = CreateObject("ADODB.Recordset")
{
   Open("Select * from Students","Provider=vfpoledb;Data Source=D:\Program
Files\Microsoft Visual Studio\Vfp98\Wizards\Template\Students And
Classes\Data\STUDENTS AND CLASSES.DBC;Collating Sequence=machine"1,1 )
}
DataSource = rs
```

If the VirtualMode property is False, the control updates automatically the item associated to the record in the control, if a change occurs in the current record. The control doesn't update the Items collection if the records are deleted or added to the table, while the DetectDelete and DetectAddNew properties are False.

Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample binds the "Employees" table in NWIND database to your control:

```
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 Grid1
   .BeginUpdate
      .AutoEdit = True
      .ColumnAutoResize = False
      .MarkSearchColumn = False
      .DrawGridLines = True
      Set .DataSource = rs
   .EndUpdate
End With
```

The following sample releases the data source that was previously bounded to the control

```
Set .DataSource = Nothing
```

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;

_Recordset21Ptr 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_grid.BeginUpdate();
            m_grid.SetColumnAutoResize( FALSE );
            m_grid.SetDataSource( spRecordset );
            m_grid.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 Grid.DefaultEditorOption(Name as EditorOptionEnum) as Variant

Specifies a default option for an editor.

Type	Description
Name as EditorOptionEnum	Specifies the name of the editor whose option is being changed.
Variant	A Variant expression that indicates the newly value for the option.

Use the DefaultEditorOption property to specify default option for the editors of a specified type. The DefaultEditorOption property affects the editors that are created after the calling the DefaultEditorOption method. The DefaultEditorOption property doesn't affect the editors being already created. Use the Option property to change the option for a particular editor. For instance, you can use the DefaultEditorOption property to localize the name of the months for all date editors using the exDateMonths option.

property Grid.DefaultItemHeight as Long

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

Type	Description
Long	A long expression that indicates the default item's height.

The DefaultItemHeight property specifies the height of the items. Changing the property fails if the control contains already items. You can change the DefaultItemHeight property at design time, or at runtime, before adding any new items to the Items collection. Use the ItemHeight property to specify the height of a specified item. Use the ScrollBySingleLine property when using the items with different heights. Use the CellSingleLine property to specify whether the cell displays the caption using multiple lines. If the column's EditType property is PictureType, the DefaultItemHeight property defines the height of the cell inside picture. In CardView mode, the DefaultItemHeight property determines the height in pixels of the title of the cards. Use the exCardViewTitleFormat option to hide the titles in CardView mode.

property Grid.Description(Type as DescriptionTypeEnum) as String

Changes descriptions for different parts in the control.

Type	Description
Type as DescriptionTypeEnum	A long expression that defines the part being changed
String	A string value that indicates the part's description.

Use the Description property to customize the captions for control filter bar window. For instance, the Description(exFilterAll) = "(Include All)" changes the "(All)" item description in the filter bar window.

The following VB sample changes the description of (All) item in the drop down filter window:

```
Grid1.Description(exFilterBarAll) = "(Toate)"
```

The following C++ sample changes the description of (All) item in the drop down filter window:

```
m_grid.SetDescription( 0  /*exFilterBarAll*/ , "(Toate)" );
```

The following VB.NET sample changes the description of (All) item in the drop down filter window:

```
With AxGrid1
   .set_Description(EXGRIDLib.DescriptionTypeEnum.exFilterBarAll, "(Toate)")
End With
```

The following C# sample changes the description of (All) item in the drop down filter window:

```
axGrid1.set_Description(EXGRIDLib.DescriptionTypeEnum.exFilterBarAll, "(Toate)");
```

The following VFP sample changes the description of (All) item in the drop down filter window:

```
thisform.Grid1.Description(0) = "(Toate)"
```

property Grid.DetectAddNew as Boolean

Specifies whether the control detects when a new record is added to the bounded recordset.

Type	Description
Boolean	A boolean expression that indicates whether the control detects when a new record is added to the bounded recordset.

By default, the DetectAddNew property is False. 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 (ADO, ADODB recordset only). The /COM version of the component provides the DataSource property, which can be used to bound the control's view to a DAO recordset. By default, once you set the DataSource property to a recordset, all changes you do on the control will be updated in the associated recordset. Because the DAO object does not provide any notifications or events the control is not able to detect any AddNew or Delete method that has been called. Instead, the control provides the AddItem and RemoveItem events that notifies your application once a new item is added to the control, or when an item is deleted. Based on these events, you will be able to manipulate the DAO recordset appropriate as in the following samples. In addition, the control fires the Error event in case any error occurs when handling the ADO or DAO recordsets, For instance, trying to update a read-only field. In conclusion, if user changes a cell/value in the control, the associated field in the recordset is automatically updated. If any error occurs on updating the associated record, the Error event is fired which describes the error.

Handling the AddNew method in the control, using the DAO recordset on MS Access

- *Insert a Button and the Control to a form, and name them as cmdAddNew and Grid1*
- *Add the Form_Load event of the form with the following code*:

```
Private Sub Form_Load()
   With Grid1
      .BeginUpdate
         .DataSource = CurrentDb.OpenRecordset("Employees")
         .DetectAddNew = True
      .EndUpdate
   End With
End Sub
```

The code binds the control to a DAO recordset. Please notice, that the DetectAddNew property is set after calling the DataSource method. Setting the DetectAddNew property on True, makes the control associate new items with new records added during the AddItem event as shown bellow.

- *Add the Click event of the cmdAddNew button with the following code*

```
Private Sub cmdAddNew_Click()
   With Grid1.Items
      .EnsureVisibleItem .AddItem
   End With
End Sub
```

The code adds a new item to the control and ensures that the new item fits the control's client area. The Items.AddItem call makes the control to fire the AddItem event, which will actually add the new record to the database, as in the following code

- *Add the AddItem event of the Control with the following code:*

```
Private Sub Grid1_AddItem(ByVal Item As Long)
   With Grid1
      If .DetectAddNew Then
         With .DataSource
            .AddNew
            !Lastname = "new"
            !FirstName = "new"
            .Update
         End With
      End If
   End With
End Sub
```

The code adds a new record to the bounded recordset. Here you need to insert or update the required fields so the new record is added to the DAO recordset. Once the event is finished, the new item is associated with the new record in the database, so from now on, any change to the item will be reflected in the recordset.

Handling the AddNew method in the control, using the ADO recordset in VB

- *Insert a Button and the Control to a form, and name them as cmdAddNew and Grid1*
- *Add the Form_Load event of the form with the following code*:

```
Private Sub Form_Load()
   With Grid1
      Set rs = CreateObject("ADOR.Recordset")
      With rs
         .Open "Employees", "Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=\sample.accdb", 3, 3
      End With
      .DataSource = rs
      .DetectAddNew = True
   End With
End Sub
```

The code binds the control to an ADO recordset.

- *Add the Click event of the cmdAddNew button with the following code*

```
Private Sub cmdAddNew_Click()
   With Grid1.DataSource
      .AddNew Array("FirstName", "LastName"), Array("new", "new")
      .Update
   End With
End Sub
```

The code adds a new record to the attached recordset, and the control will add a new associated item, because the DetectAddNew method is True.

property Grid.DetectDelete as Boolean

Specifies whether the control detects when a record is deleted from the bounded recordset.

Type	Description
Boolean	A boolean expression that indicates whether the control detects when a record is deleted from the bounded recordset.

By default, the DetectDelete property is False. If the DetectDelete property is True, the control is notified when a record is deleted, and the associated item is removed from the control's items collection (valid for the ADO, ADODB recordsets). The /COM version of the component provides the DataSource property, which can be used to bound the control's view to a DAO recordset. By default, once you set the DataSource property to a recordset, all changes you do on the control will be updated in the associated recordset. Because the DAO object does not provide any notifications or events the control is not able to detect any AddNew or Delete method that has been called. Instead, the control provides the AddItem and RemoveItem events that notifies your application once a new item is added to the control, or when an item is deleted. Based on these events, you will be able to manipulate the DAO recordset appropriate as in the following samples. In addition, the control fires the Error event in case any error occurs when handling the ADO or DAO recordsets, For instance, trying to update a read-only field. In conclusion, if user changes a cell/value in the control, the associated field in the recordset is automatically updated. If any error occurs on updating the associated record, the Error event is fired which describes the error.

Handling the Delete method in the control, using the DAO recordset on MS Access

- *Insert a Button and the Control to a form, and name them as cmdDelete and Grid1*
- *Add the Form_Load event of the form with the following code:*

```
Private Sub Form_Load()
   With Grid1
      .BeginUpdate
         .DataSource = CurrentDb.OpenRecordset("Employees")
         .DetectDelete = True
      .EndUpdate
   End With
End Sub
```

The code binds the control to a DAO recordset. The DetectDelete property on True, makes the control to move the current record on the item to be deleted, and to remove

any reference to the record to be deleted.

- *Add the Click event of the cmdDelete button with the following code*

```
Private Sub cmdRemove_Click()
   With Grid1.Items
      .RemoveItem .FocusItem
   End With
End Sub
```

The code removes the focused item. The Items.RemoveItem call makes the control to fire the RemoveItem event, which will actually delete the associated record in the database, as in the following code

- *Add the RemoveItem event of the Control with the following code:*

```
Private Sub Grid1_RemoveItem(ByVal Item As Long)
   With Grid1
      If .DetectDelete Then
         With .DataSource
            .Delete
         End With
      End If
   End With
End Sub
```

The code deletes the current record.

Handling the Delete method in the control, using the ADO recordset in VB

- *Insert a Button and the Control to a form, and name them as cmdRemove and Grid1*
- *Add the Form_Load event of the form with the following code:*

```
Private Sub Form_Load()
   With Grid1
      Set rs = CreateObject("ADOR.Recordset")
      With rs
         .Open "Employees", "Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=\sample.accdb", 3, 3
      End With
```

```
        .DataSource = rs
        .DetectDelete = True
    End With
End Sub
```

The code binds the control to an ADO recordset.

- *Add the Click event of the cmdDelete button with the following code*

```
Private Sub cmdRemove_Click()
    With Grid1.DataSource
        .Delete
    End With
End Sub
```

The Delete method of the recordset removes the current record (select a new item to the control, and the current record is changed), and due DetectDelete the associated item is removed from the view.

method Grid.DiscardValidateValue ()

Cancels the current validation process, and restores back the modified cells.

Type	Description

The DiscardValidateValue method has effect only if the CauseValidateValue property is not zero. The DiscardValidateValue method restores the values for modified cell during the validation. For instance, pressing the Cancel button during the ValidateValue event can restore the values for modified cells, using the DiscardValidateValue method. The DiscardValidateValue automatically closes the current editor. The EditClose method can be used to programmatically closes the focused editor.

property Grid.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. The GridLineStyle property to specify the style for horizontal or/and vertical gridlines in the control. 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.

method Grid.Edit ([Options as Variant])

Edits the focused cell.

Type	Description
Options as Variant	Optional. If missing, the control edits the focused cell. A long expression that indicates the handle of a locked item. Use the LockedItem property to retrieve the handle of a locked/fixed item.

The Edit method starts editing the focused cell, if the cell has an editor assigned. Use the Editor property of the Column object, or CellEditor property to assign an editor to a cell. The focused cell is determined by the intersection of the focused item and the focused column. Use the FocusItem property to get the handle of the focused item. Use the FocusColumnIndex property to determine the index of the focused column. The control fires the Edit event when the edit operation is about to start. The edit operation doesn't start if the control's ReadOnly property is True, or if the cell's editor is hidden (CellEditorVisible property is False). Use the Editing property to check whether the control is in edit mode, or to get the window's handle for the built-in editor that's visible and focused. The EditClose method closes the current editor. Use the ValidateValue event to validate the values that user enters.

The edit events are fired in the following order:

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

If the Options parameter is missing, the control edits the focused cell. The FocusItem and FocusColumnIndex properties indicates the focused cell. If the Options parameter is present, the control edits the item that Options parameter indicates.

For instance, the following VB sample edits a locked item when the user clicks a cell:

```
Private Sub Grid1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
  With Grid1
    If Button = 1 Then
```

```
        Dim h As EXGRIDLibCtl.HITEM, c As Long, hit As EXGRIDLibCtl.HitTestInfoEnum
        h = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
        If Not h = 0 Then
          If (.Items.IsItemLocked(h)) Then
            .FocusColumnIndex = c
            .Edit h
          End If
        End If
      End If
    End With
End Sub
```

Call the DoEvents (Processes all Windows messages currently in the message queue) method after Edit method to start immediately the edit operation.

The following C++ sample edits edits a locked item when the user clicks a cell:

```
void OnMouseUpGrid1(short Button, short Shift, long X, long Y)
{
  CItems items = m_grid.GetItems();
  long c = 0, hit = 0, h = m_grid.GetItemFromPoint( X, Y, &c, &hit);
  if ( h != 0 )
    if ( items.GetIsItemLocked( h ) )
    {
      m_grid.SetFocusColumnIndex( c );
      m_grid.Edit( COleVariant( h ) );
    }
}
```

The following VB.NET sample edits edits a locked item when the user clicks a cell:

```
Private Sub AxGrid1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_MouseUpEvent) Handles AxGrid1.MouseUpEvent
  With AxGrid1
    Dim i As Integer, c As Integer, hit As EXGRIDLib.HitTestInfoEnum
    i = .get_ItemFromPoint(e.x, e.y, c, hit)
    If (Not (i = 0)) Then
      If (.Items.IsItemLocked(i)) Then
        .FocusColumnIndex = c
```

```
            .Edit(i)
        End If
    End If
  End With
End Sub
```

The following C# sample edits edits a locked item when the user clicks a cell:

```
private void axGrid1_MouseUpEvent(object sender,
AxEXGRIDLib._IGridEvents_MouseUpEvent e)
{
  int c = 0;
  EXGRIDLib.HitTestInfoEnum hit;
  int i = axGrid1.get_ItemFromPoint(e.x, e.y, out c, out hit);
  if (i != 0)
  {
    if ( axGrid1.Items.get_IsItemLocked( i ) )
      {
        axGrid1.FocusColumnIndex = c;
        axGrid1.Edit(i);
      }
  }
}
```

The following VFP sample edits edits a locked item when the user clicks a cell:

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

local c, hit
c = 0
hit = 0
with thisform.Grid1
  .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
  if ( .Items.DefaultItem <> 0 )
    if ( .Items.IsItemLocked(0) )
      .FocusColumnIndex = c
      .Object.Edit(.Items.DefaultItem)
```

```
      endif
    endif
endwith
```

method Grid.EditClose ()

Closes the current editor, if the control runs in the edit mode.

Type	Description

Use the EditClose method to close programmatically the current editor. Use the Edit method to start editing a cell. Use the Editing property to check whether the control runs in the edit mode.

The following VB sample closes the editor when user hits the enter key:

```
Private Sub Grid1_KeyDown(KeyCode As Integer, Shift As Integer)
   With Grid1
      If Not (.Editing = 0) Then
         If (KeyCode = vbKeyReturn) Then
            .EditClose
            KeyCode = 0
         End If
      End If
   End With
End Sub
```

The following C++ sample closes the editor when user hits the enter key:

```
void OnKeyDownGrid1(short FAR* KeyCode, short Shift)
{
   if ( *KeyCode == VK_RETURN )
      if ( m_grid.GetEditing() != 0 )
      {
         m_grid.EditClose();
         *KeyCode = 0;
      }
}
```

The following C# sample closes the editor when user hits the enter key:

```
private void axGrid1_KeyDownEvent(object sender,
AxEXGRIDLib._IGridEvents_KeyDownEvent e)
{
```

```
    if (Convert.ToUInt32(e.keyCode) == Convert.ToUInt32(Keys.Enter))
        if (axGrid1.Editing != 0)
        {
            axGrid1.EditClose();
            e.keyCode = 0;
        }
}
```

The following VB.NET sample closes the editor when user hits the enter key:

```
Private Sub AxGrid1_KeyDownEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_KeyDownEvent) Handles AxGrid1.KeyDownEvent
    If (Convert.ToUInt32(e.keyCode) = Convert.ToUInt32(Keys.Enter)) Then
        With AxGrid1
            If Not (.Editing = 0) Then
                .EditClose()
                e.keyCode = 0
            End If
        End With
    End If
End Sub
```

The following VFP sample closes the editor when user hits the enter key:

```
*** ActiveX Control Event ***
LPARAMETERS keycode, shift

if ( keycode = 13 ) &&vkReturn
    with thisform.Grid1.Object
        if ( .Editing() != 0 )
            .EditClose()
            keycode = 0
        endif
    endwith
endif
```

property Grid.Editing as Long

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

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

Use the Editing property to check whether the control is in edit mode. Use the Editing property to get the window's handle for the built-in editor while editing. Use the Edit method to start editing the focused cell. Use the EditType property to define the column's editor. Use the ReadOnly property to make the control read only. Call the EditClose method to close the current editor. The EditingText property returns the caption being shown on the editor while the control runs in edit mode. The Editing property returns a not-zero value only if called during the EditOpen, Change or EditClose event.

The edit events are fired in the following order:

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

The following VB sample closes the current editor if the user presses the enter key:

```
Private Sub Grid1_KeyDown(KeyCode As Integer, Shift As Integer)
  With Grid1
    If Not (.Editing = 0) Then
      If (KeyCode = vbKeyReturn) Then
        .EditClose
        KeyCode = 0
      End If
    End If
  End With
End Sub
```

The following C++ sample closes the editor when user hits the enter key:

```
void OnKeyDownGrid1(short FAR* KeyCode, short Shift)
{
   if ( *KeyCode == VK_RETURN )
      if ( m_grid.GetEditing() != 0 )
      {
         m_grid.EditClose();
         *KeyCode = 0;
      }
}
```

The following C# sample closes the editor when user hits the enter key:

```
private void axGrid1_KeyDownEvent(object sender,
AxEXGRIDLib._IGridEvents_KeyDownEvent e)
{
   if (Convert.ToUInt32(e.keyCode) == Convert.ToUInt32(Keys.Enter))
      if (axGrid1.Editing != 0)
      {
         axGrid1.EditClose();
         e.keyCode = 0;
      }
}
```

The following VB.NET sample closes the editor when user hits the enter key:

```
Private Sub AxGrid1_KeyDownEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_KeyDownEvent) Handles AxGrid1.KeyDownEvent
   If (Convert.ToUInt32(e.keyCode) = Convert.ToUInt32(Keys.Enter)) Then
      With AxGrid1
         If Not (.Editing = 0) Then
            .EditClose()
            e.keyCode = 0
         End If
      End With
   End If
End Sub
```

The following VFP sample closes the editor when user hits the enter key:

```
*** ActiveX Control Event ***
LPARAMETERS keycode, shift

if ( keycode = 13 ) &&vkReturn
   with thisform.Grid1.Object
      if ( .Editing() != 0 )
         .EditClose()
         keycode = 0
      endif
   endwith
endif
```

If your application still requires the string that user types into an text box inside the exGrid control. you can use the following VB trick:

```
Private Sub Grid1_Change(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As Long,
NewValue As Variant)
   ' Finds the text inside the text box, in case that NewValue parameter is changed to a
valid data
   Debug.Print getWndText(getEditWnd(Grid1))
End Sub

Private Function getEditWnd(ByVal g As EXGRIDLibCtl.Grid) As Long
   Dim h As Long
   h = GetWindow(g.hwnd, GW_CHILD)
   While Not (h = 0)
      If (getWndClass(h) = "HolderBuiltIn") Then
            getEditWnd = GetWindow(h, GW_CHILD)
         Exit Function
      End If
      h = GetWindow(h, GW_HWNDNEXT)
   Wend
   getEditWnd = 0
End Function

Private Function getWndText(ByVal h As Long) As String
```

```
    Dim s As String
    s = Space(1024)
    GetWindowText h, s, 1024
    getWndText = To0(s)
End Function

Private Function getWndClass(ByVal h As Long) As String
    Dim s As String
    s = Space(1024)
    GetClassName h, s, 1024
    getWndClass = To0(s)
End Function

Private Function To0(ByVal s As String) As String
    To0 = Left$(s, InStr(s, Chr$(0)) - 1)
End Function
```

The sample requires the following API declarations:

```
Private Declare Function GetWindow Lib "user32" (ByVal hwnd As Long, ByVal wCmd As Long) As Long
Private Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" (ByVal hwnd As Long, ByVal lpString As String, ByVal cch As Long) As Long
Private Declare Function GetClassName Lib "user32" Alias "GetClassNameA" (ByVal hwnd As Long, ByVal lpClassName As String, ByVal nMaxCount As Long) As Long
Private Const GW_CHILD = 5
Private Const GW_HWNDNEXT = 2
```

The following C++ sample displays a message box with the caption that user types inside the text box of an editor:

```
HWND getEditWnd( HWND h )
{
    TCHAR szName[1024] = _T("");
    h = GetWindow( h, GW_CHILD );
    while ( !( h == 0 ) )
    {
```

```
        GetClassName( h, szName, 1024 );
        if ( _tcscmp( _T("HolderBuiltIn"), szName ) == 0 )
            return GetWindow( h, GW_CHILD );
        h = GetWindow( h, GW_HWNDNEXT );
    }
    return 0;
}

void OnChangeGrid1(long Item, long ColIndex, VARIANT FAR* NewValue)
{
    HWND h = getEditWnd( m_grid.m_hWnd );
    if ( h )
    {
        TCHAR szText[1024] = _T("");
        ::GetWindowText( h, szText, 1024 );
        ::MessageBox( NULL, szText,NULL, NULL );
    }
}
```

property Grid.EditingText as String

Specifies the caption of the editor during editing.

Type	Description
String	A String expression that specifies the caption of the field during editing mode.

By default, the EditingText property is "". The EditingText property returns the caption being shown on the editor while the control runs in edit mode. The control is in edit mode, if the Editing property returns a not-zero value. The EditingText property has effect only if called during the EditOpen, Change or EditClose event.

The edit events are fired in the following order:

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

property Grid.Enabled as Boolean

Enables or disables the control.

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

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

Ends recording the UI operations and adds the undo/redo operations as a block, so they all can be restored at once, if Undo method is performed.

Type	Description

The StartBlockUndoRedo method starts recording the UI operations as a block on undo/redo operations The method has effect only if the AllowUndoRedo property is True. The EndBlockUndoRedo method collects all undo/redo operations since StartBlockUndoRedo method was called and add them to the undo/redo queue as a block. This way the next call on a Undo operation, the entire block is restored, so all UI operations are restored. The EndBlockUndoRedo method must be called the same number of times as the StartBlockUndoRedo method was called. For instance, if you have called the StartBlockUndoRedo twice the EndBlockUndoRedo method must be called twice too, and the collected operations are added to the control's queue of undo/redo operations at the end.

method Grid.EndUpdate ()

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

Type	Description

Use BeginUpdate and EndUpdate statement each time when the control requires more changes. Using the BeginUpdate and EndUpdate methods increase the speed of changing the control properties by preventing it from painting during changing.

The following VB sample prevents the control from painting while it adds one column, and one item, using the BeginUpdate and EndUpdate methods:

```
With Grid1
  .BeginUpdate
    .LinesAtRoot = LinesAtRootEnum.exLinesAtRoot
    .FullRowSelect = False
    .DefaultItemHeight = 24
    With .Columns.Add("Mask")
      With .Editor
        .EditType = EditTypeEnum.MaskType
        .Appearance = SingleApp
        .Mask = "{0,255}\.{0,255}\.{0,255}\.{0,255}"
      End With
    End With
    .Items.AddItem "193.226.40.161"
  .EndUpdate
End With
```

The following C++ sample prevents the control from painting while adding columns and items:

```
#include "Column.h"
#include "Columns.h"
m_grid.BeginUpdate();
m_grid.SetColumnAutoResize( FALSE );
CColumns columns = m_grid.GetColumns();
for ( long i = 0; i < 4; i++ )
  columns.Add( "NewColumn" );
CItems items = m_grid.GetItems();
```

```
for ( long j = 0; j < 10; j++ )
{
   COleVariant vtItem( items.AddItem( COleVariant( "new item " ) ) );
   for ( long k = 1 ; k < 4; k++ )
      items.SetCellValue( vtItem, COleVariant( k ), COleVariant( "new item " ) );
}
m_grid.EndUpdate();
```

The following VB.NET sample prevents the control from painting while adding columns and items:

```
With AxGrid1
   .BeginUpdate()
   .LinesAtRoot = EXGRIDLib.LinesAtRootEnum.exLinesAtRoot
   .FullRowSelect = False
   .DefaultItemHeight = 24
   With .Columns.Add("Mask")
      With .Editor
         .EditType = EXGRIDLib.EditTypeEnum.MaskType
         .Appearance = EXGRIDLib.AppearanceEnum.Etched
         .Mask = "{0,255}\.{0,255}\.{0,255}\.{0,255}"
      End With
   End With
   .Items.AddItem("193.226.40.161")
   .EndUpdate()
End With
```

The following C# sample prevents the control from painting while adding columns and items:

```
axGrid1.BeginUpdate();
EXGRIDLib.Columns columns = axGrid1.Columns;
columns.Add("Column 1");
columns.Add("Column 3");
EXGRIDLib.Items items = axGrid1.Items;
items.AddItem("new item");
items.AddItem("new item");
axGrid1.EndUpdate();
```

The following VFP sample prevents the control from painting while adding columns and items:

```
with thisform.Grid1
   .BeginUpdate
      .LinesAtRoot = .t.
      .FullRowSelect = .f.
      .DefaultItemHeight = 24
      With .Columns.Add("Mask")
         With .Editor
            .EditType = 8
            .Appearance = 1
            .Mask = "{0,255}\.{0,255}\.{0,255}\.{0,255}"
         EndWith
      EndWith
      .Items.AddItem("193.226.40.161")
   .EndUpdate
endwith
```

property Grid.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 Grid1_BeforeExpandItem(ByVal Item As EXGRIDLibCtl.HITEM, Cancel As
Variant)
  With Grid1
    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 Grid.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 column's index being scrolled, or a string expression that indicates the column's caption or the column's key.

This method ensures that a column is at least partially visible. The control scrolls the content if necessary. The control automatically calls EnsureVisibleColumn method when the user clicks a cell in the column. Use the EnsureVisibleItem method to ensure that a specified 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.

The following VB sample changes the searching column when user clicks a cell:

```
Private Sub Grid1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As
Single)
  With Grid1
    Dim c As Long, hit as Long
    Dim h As HITEM
    h = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
    If Not (h = 0) Then
      .SearchColumnIndex = c
    End If
  End With
End Sub
```

The following VB sample ensures that a hidden column is going visible, and fits the control's client area:

```
With Grid1
  .Columns("G").Visible = True
  DoEvents
  .EnsureVisibleColumn ("G")
End With
```

The following C++ sample ensures that a hidden column is going visible, and fits the control's client area:

```
COleVariant vtColumn("G");
m_grid.GetColumns().GetItem( vtColumn ).SetVisible( TRUE );
DoEvents();
m_grid.EnsureVisibleColumn( vtColumn );
```

where an equivalent of DoEvents in C++ is:

```
void DoEvents()
{
   MSG m = {0};
   while ( PeekMessage( &m, NULL, NULL, NULL, PM_REMOVE ) )
   {
      TranslateMessage( &m );
      DispatchMessage( &m );
   }
}
```

Calling the DoEvents method it not required if before you are not changing the column's layout. Making a column visible or hidden changes the arrangement of the columns, that's processed later. The DoEvents method forces the control to arrange the column, before calling the EnsureVisibleColumn method.

property Grid.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 Grid.ExecuteTemplate (Template as String)

Executes a template and returns the result.

Type	Description
Template as String	A Template string being executed

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

Use the ExecuteTemplate property to returns the result of executing a template file. Use the Template property to execute a template without returning any result. Use the ExecuteTemplate property to execute code by passing instructions as a string (template string). For instance, you can use the EXPRINT.PrintExt = CONTROL.ExecuteTemplate("me") to print the control's content.

For instance, the following sample retrieves the the handle of the first visible item:

```
Debug.Print Grid1.ExecuteTemplate("Items.FirstVisibleItem()")
```

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

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

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

The Template script is composed by lines of instructions. Instructions are separated by

"\n\r" (newline) characters.

An instruction can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable **=** property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))*
- property**(** list of arguments **)** = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method**(** list of arguments **)** *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- object**.** property(list of arguments)**.**property(list of arguments)…. *The .(dot) character splits the object from its property. For instance, the Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The Template supports the following general functions:

- **RGB(**R,G,B**)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **CreateObject(**progID**)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

property Grid.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. 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. In CardView mode, the ExpandOnDblClick property specifies whether a card is expanded or collapsed when a card is double clicked.

property Grid.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. In CardView mode, the ExpandOnKeys property allows expanding or collapsing the cards using the + or - keys on the numeric keypad.

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 Grid1_KeyDown(KeyCode As Integer, Shift As Integer)
  With Grid1.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 OnKeyDownGrid1(short FAR* KeyCode, short Shift)
{
```

```
    CItems items = m_grid.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 AxGrid1_KeyDownEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_KeyDownEvent) Handles AxGrid1.KeyDownEvent
    Select Case (e.keyCode)
        Case Keys.Add
            With AxGrid1.Items
                .ExpandItem(.FocusItem) = True
            End With
        Case Keys.Subtract
            With AxGrid1.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 axGrid1_KeyDownEvent(object sender,
AxEXGRIDLib._IGridEvents_KeyDownEvent e)
{
    if ( ( e.keyCode == Convert.ToInt16(Keys.Add)  ) || (e.keyCode ==
Convert.ToInt16(Keys.Subtract) ) )
```

```
    axGrid1.Items.set_ExpandItem( axGrid1.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.Grid1.Items
   if ( keycode = 107 )
      .DefaultItem = .FocusItem
      .ExpandItem(0) = .t.
   else
      if ( keycode = 109 )
         .ExpandItem(0) = .f.
      endif
   endif
endwith
```

property Grid.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. By default, the ExpandOnSearch property is False. 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. Use the exExpandOnSearch type to expand items while user types characters in a drop down editor. 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 Grid.Export ([Destination as Variant], [Options as Variant])

Exports the control's data to a CSV or HTML format.

Type	Description
Destination as Variant	A String expression that specifies the file/format to be created. The Destination parameter indicates the format to be created as follows: • if "htm" or "html", the control returns the HTML format (including CSS style) • Any file-name that ends on ".htm" or ".html" creates the file with the HTML format inside • missing, empty, or any other case the Export exports the control's data in CSV format. No error occurs, if the Export method can not create the file.
Options as Variant	A String expression that specifies the options to be used when exporting the control's data, as explained bellow.

Return	Description
Variant	A String expression that indicates the format being exported. It could be CSV or HTML format based on the Destination parameter.

The Export method can export the control's DATA to a CSV or HTML format. The Export method can export a collection of columns from selected, visible, check or all items. By default, the control export all items, unless there is no filter applied on the control, where only visible items are exported. No visual appearance is saved in CSV format, instead the HTML format includes the visual appearance in CSS style.

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:

export.txt - Microsoft Excel

FILE HOME INSERT PAGE LAYOUT FORMULAS DATA REVIEW VIEW ADD-INS TEAM Sign in

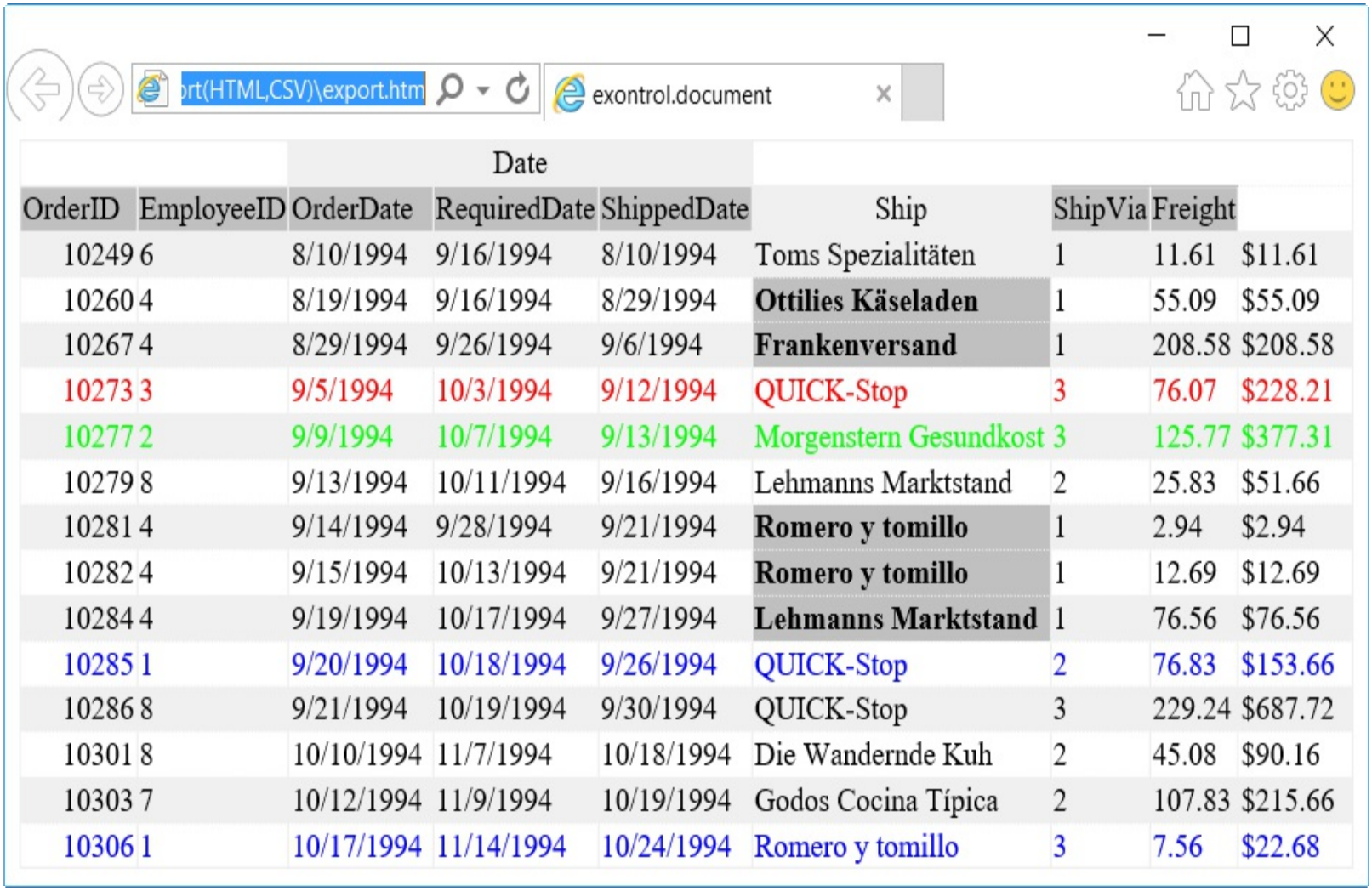
A1

	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

export

READY 100%

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.

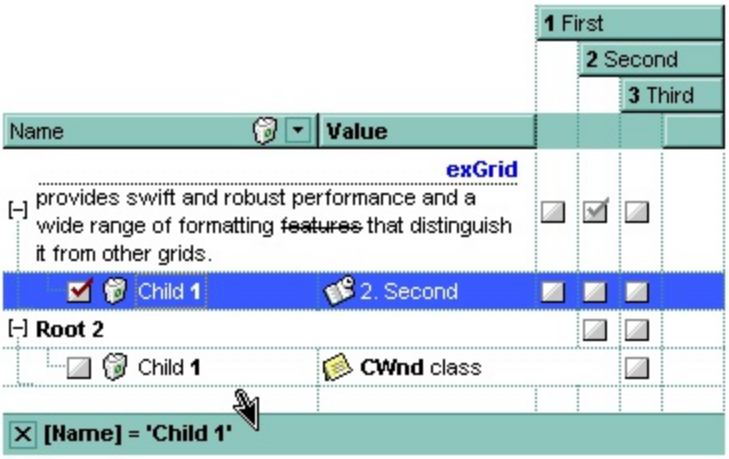
You can use the Copy/CopyTo to export the control's view to clipboard/EMF/BMP/JPG/PNG/GIF or PDF format.

property Grid.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.

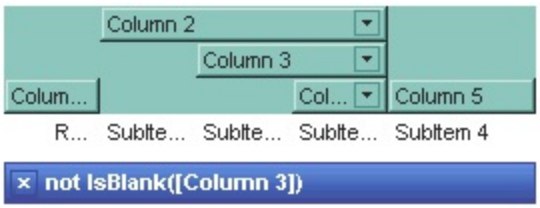
Use the FilterBarForeColor and FilterBarBackColor properties to define the colors used to paint the description for control's filter. Use the FilterBarHeight property to hide the control's filter bar header. Use the BackColor property to specify the control's background color. Use the BackColorLevelHeader property to specify the background color of the header when it displays multiple levels.



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 Grid1
   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_grid.GetVisualAppearance().Add( 0x2,
COleVariant(_T("D:\\Temp\\ExGrid.Help\\fbarclose.ebn")) );
m_grid.GetVisualAppearance().Add( 0x12,
COleVariant(_T("D:\\Temp\\ExGrid.Help\\filterbar.ebn")) );
m_grid.SetBackground( 1 /*exFooterFilterBarButton*/, 0x2000000 );
m_grid.SetFilterBarBackColor( 0x12000000 );
m_grid.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 AxGrid1
   With .VisualAppearance
      .Add(&H2, "D:\Temp\ExGrid.Help\fbarclose.ebn")
      .Add(&H12, "D:\Temp\ExGrid.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:

```
axGrid1.VisualAppearance.Add(0x2, "D:\\Temp\\ExGrid.Help\\fbarclose.ebn");
axGrid1.VisualAppearance.Add(0x12, "D:\\Temp\\ExGrid.Help\\filterbar.ebn");
axGrid1.set_Background(EXTREELib.BackgroundPartEnum.exFooterFilterBarButton,
0x2000000);
```

```
axGrid1.Template = "FilterBarBackColor = 301989888";
axGrid1.FilterBarForeColor = Color.White;
```

The following VFP sample changes the visual appearance for the "close" button in the control's filter bar:

```
With thisform.Grid1
   With .VisualAppearance
      .Add(2, "D:\Temp\ExGrid.Help\fbarclose.ebn")
      .Add(18, "D:\Temp\ExGrid.Help\filterbar.ebn")
   EndWith
   .Object.Background(1) = 33554432
   .FilterBarBackColor = 301989888
   .FilterBarForeColor = RGB(255,255,255)
EndWith
```

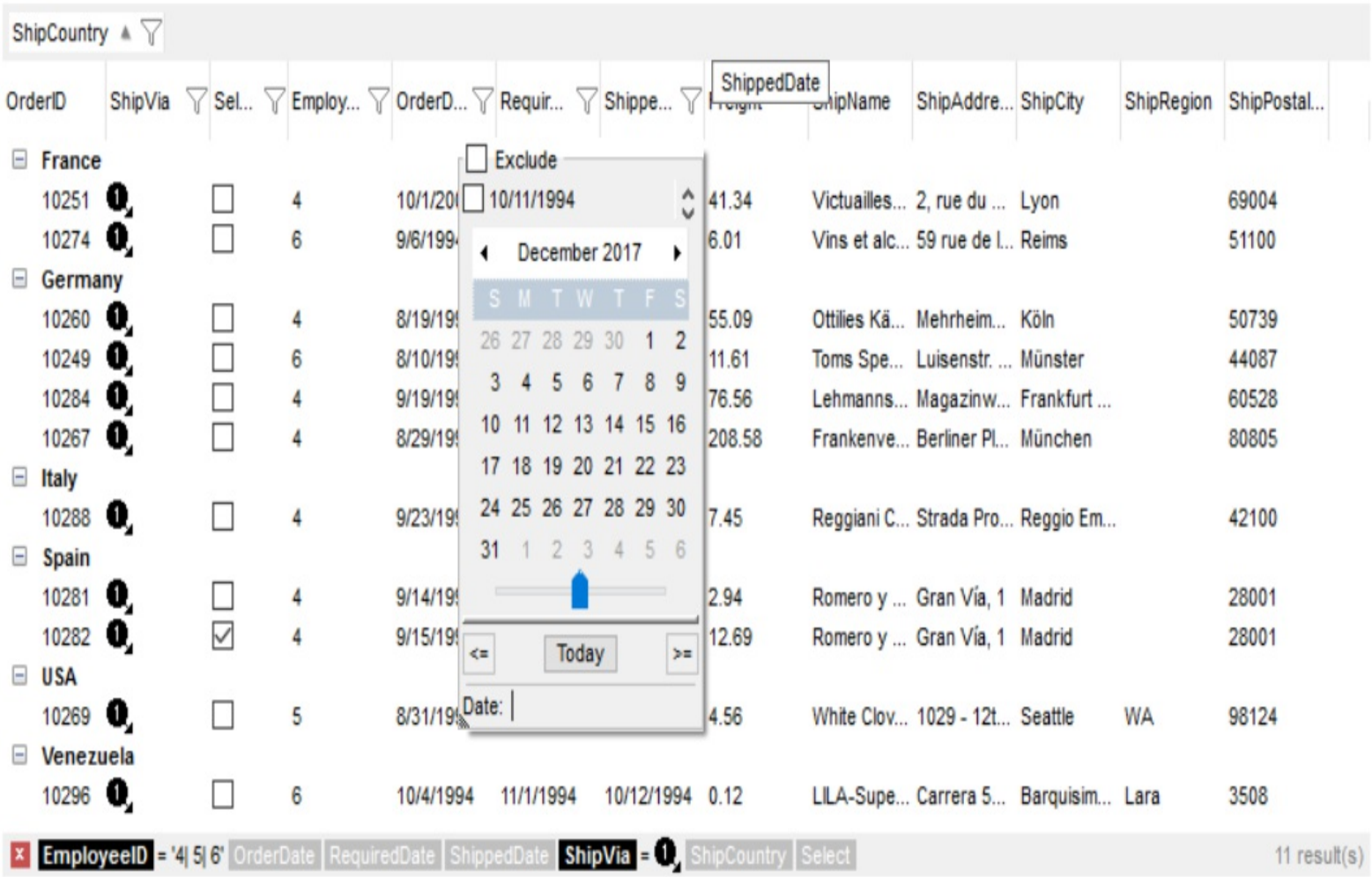
The 301989888 value is the 0x12000000 value in hexadecimal.

property Grid.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

no filter

- "" displays no filter bar, if no filter is applied, else it displays the current filter

[EmployeeID] = '4| 5| 6' and [ShipVia] =

- "`<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

[EmployeeID] = '4| 5| 6' and [ShipVia] =

- "value replace ` and ` with `<fgcolor=FF0000> and </fgcolor>`", replace the AND keyword with a different foreground color

[EmployeeID] = '4| 5| 6' and [ShipVia] =

- "value replace ` and ` with `<off 4> and </off>` replace `|` with ` <off 4>or</off> ` replace ` ` with ` `", replaces the AND and | values

[EmployeeID] = '4 or 5 or 6' and [ShipVia] =

- "value replace `[` with `<bgcolor=000000><fgcolor=FFFFFF><b> ` replace `]` with ` </b></bgcolor></fgcolor>`", highlights the columns being filtered with a different background/foreground colors.

EmployeeID = '4| 5| 6' and ShipVia =

- "value + ` ` + available", displays the current filter, including all available columns to be filtered

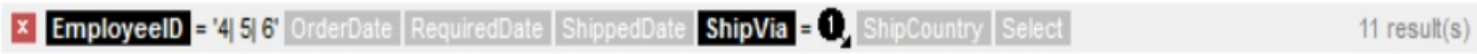
[EmployeeID] = '4| 5| 6' and [ShipVia] = [OrderDate] [RequiredDate] [ShippedDate] [ShipCountry] [Select]

- "allui" displays all available columns

[EmployeeID] = '4| 5| 6' [OrderDate] [RequiredDate] [ShippedDate] [ShipVia] = [ShipCountry] [Select]

- "((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 ^{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 Grid.FilterBarDropDownHeight as Double

Specifies the height of the drop down filter window proportionally with the height of the control's list.

Type	Description
Double	A double expression that indicates the height of the drop down filter window.

Use the FilterBarDropDownHeight property to specify the height of the drop down window filter window. 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 FilterBarDropDownWidth property to specify the width of the drop down filter window. Use the DisplayFilterButton property to display a filter button to the column's caption. Use the ShowFilter method to show programmatically the column's drop down filter window.

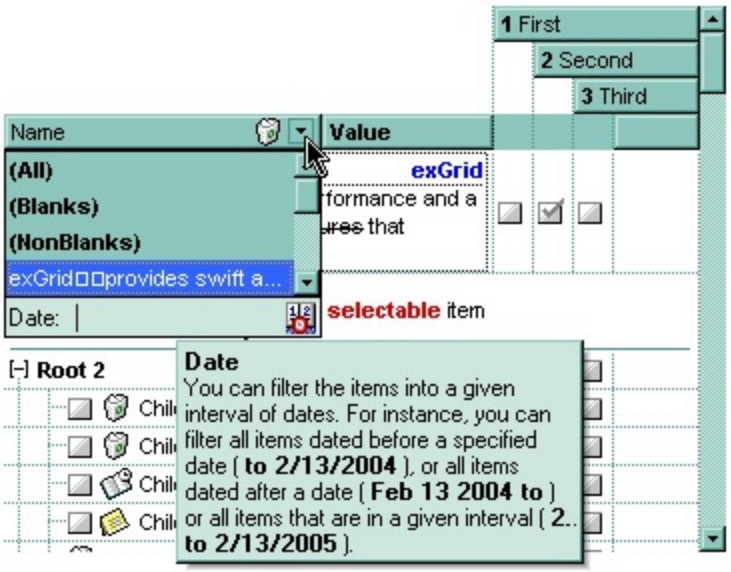
If the FilterBarDropDownHeight property is negative, the absolute value of the FilterBarDropDownHeight property indicates the height of the drop down filter window in pixels. In this case, the height of the drop down filter window is not proportionally with the height of the control's list area. For instance, the following sample specifies the height of the drop down filter window being 100 pixels:

```
With Grid1
   .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 Grid1
   .FilterBarDropDownHeight = 1
End With
```

The drop down filter window always include an item.


1 First
2 Second
3 Third
Name
Value
(All)
(Blanks)
(NonBlanks)
exGrid□□provides swift a...
Date:
exGrid
formance and a
ures that
selectable item
[-] Root 2
Date
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.. to 2/13/2005).

property Grid.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

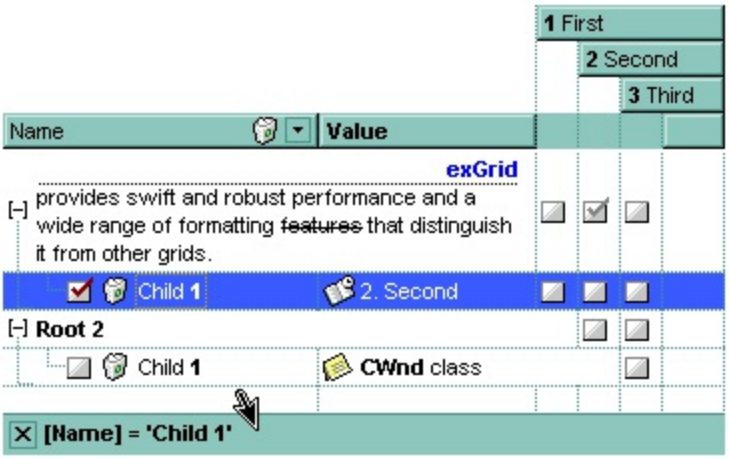
Use the FilterBarFont property to specify the font for the control's filter bar object. Use the Font property to set the control's font. Use the FilterBarHeight property to specify the height of the filter bar. Use the FilterBarCaption property to define the control's filter bar caption. Use the Refresh method to refresh the control.

property Grid.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 Grid.FilterBarHeight as Long

Specifies the height of the control's filter bar 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 ShowFilter method to show programmatically the column's drop down filter window.

property Grid.FilterBarPrompt as String

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

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

By default, the FilterBarPrompt property is "<fgcolor=808080>Start Filter...</fgcolor> ". The FilterBarPromptPattern property specifies the pattern to filter the list using the filter prompt feature. Changing the FilterBarPrompt property won't change the current filter. The FilterBarPromptColumns property specifies the list of columns to be used when filtering by prompt. The DisplayFilterButton property specifies whether the column's header displays a filter button. The VisibleItemCount property retrieves the number of visible items in the list. The control fires the FilterChanging event just before applying the filter, and FilterChange once the list gets filtered. Use the FilterBarCaption property to change the caption in the filter bar once a new filter is applied. The FilterBarFont property specifies the font to be used in the filter bar. The FilterBarBackColor property specifies the background color or the visual aspect of the control's filter bar. The FilterBarForeColor property specifies the foreground color or the control's filter bar.

The FilterBarPrompt property supports HTML format as described here:

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

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

  - exp, stores the plain text to be shown once the user clicks the anchor, such as "<a ;exp=show lines>"
  - e64, encodes in BASE64 the HTML text to be shown once the user clicks the

anchor, such as "<a ;e64=gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABu " that displays show lines- in gray when the user clicks the + anchor. The "gA8ABmABnABjABvABshIAOQAEAAHAAGESikWio+ABzABohp3iELABpABuABljY string encodes the "<fgcolor 808080>show lines<a>-</fgcolor>" The Decode64Text/Encode64Text methods of the eXPrint can be used to decode/encode e64 fields.

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

- ** ... ** displays portions of text with a different font and/or different size. For instance, the "bit" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "bit" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or <fgcolor=rrggbb> ... </fgcolor> displays text with a specified foreground color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or <bgcolor=rrggbb> ... </bgcolor> displays text with a specified background color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or <solidline=rrggbb> ... </solidline> draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or <dotline=rrggbb> ... </dotline> draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>).
- **<r>** right aligns the text
- **<c>** centers the text
- **
** forces a line-break
- **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part

of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.

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

gradient-center

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

outlined

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

color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<**sha**>shadow</**sha**>" generates the following picture:

shadow

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

outline anti-aliasing

The FilterBarPrompt property has effect only if:

- FilterBarPromptVisible property is True
- FilterBarPromptPattern property is Empty.

property Grid.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 Grid.FilterBarPromptPattern as String

Specifies the pattern for the filter prompt.

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

By default, the FilterBarPromptPattern property is empty. If the FilterBarPromptPattern property is empty, the filter bar displays the FilterBarPrompt property, if the FilterBarPromptVisible property is True. The FilterBarPromptPattern property indicates the patter to filter the list. The pattern may include wild characters if the FilterBarPromptType property is exFilterPromptPattern. The FilterBarPromptColumns specifies the list of columns to be used when filtering. Changing the FilterBarPromptPattern property does not require calling the ApplyFilter method to apply the new filter, only if filtering is required right a way.

property Grid.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

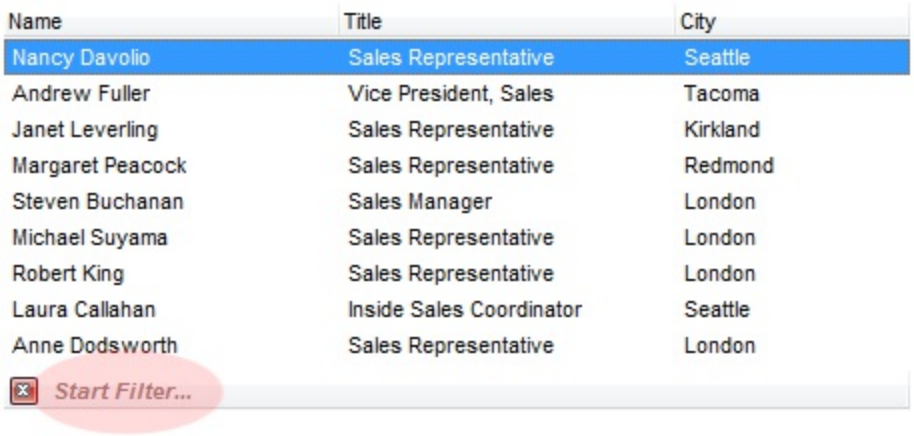
property Grid.FilterBarPromptVisible as FilterBarVisibleEnum

Shows or hides the control's filter bar including filter prompt.

Type	Description
FilterBarVisibleEnum	A FilterBarVisibleEnum expression that defines the way the control's filter bar is shown.

By default, The FilterBarPromptVisible property is exFilterBarHidden. The filter prompt feature allows you to filter the items as you type while the filter bar is visible on the bottom part of the list area. The Filter prompt feature allows at runtime filtering data on hidden columns too. Use the FilterBarPromptVisible property to show the filter prompt. Use the FilterBarPrompt property to specify the HTML caption being displayed in the filter bar when the filter pattern is missing. The FilterBarPromptPattern property specifies the pattern to filter the list. Changing the FilterBarPromptPattern property does not require calling the ApplyFilter method to apply the new filter, only if filtering is required right a way. The FilterBarCaption property defines the caption to be displayed on the control's filter bar. The FilterBarPromptType property specifies the type of filtering when the user edits the prompt in the filter bar. The FilterBarPromptColumns property specifies the list of columns to be used when filtering by prompt. The DisplayFilterButton property specifies whether the column's header displays a filter button. The VisibleItemCount property retrieves the number of visible items in the list. The control fires the FilterChanging event just before applying the filter, and FilterChange once the list gets filtered.

The following screen show shows the filter prompt:



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

Name	Title	City
Steven Buchanan	Sales Manager	London
Michael Suyama	Sales Representative	London
Robert King	Sales Representative	London
Anne Dodsworth	Sales Representative	London

london

property Grid.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 grided 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 grided 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. Use the CustomFilter property to define you custom filters.

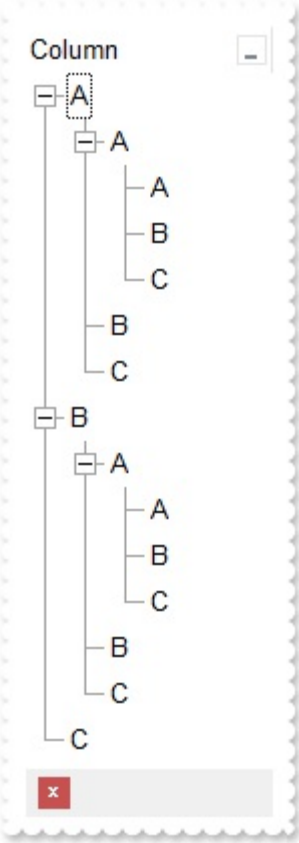
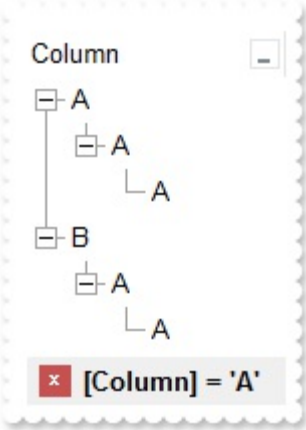
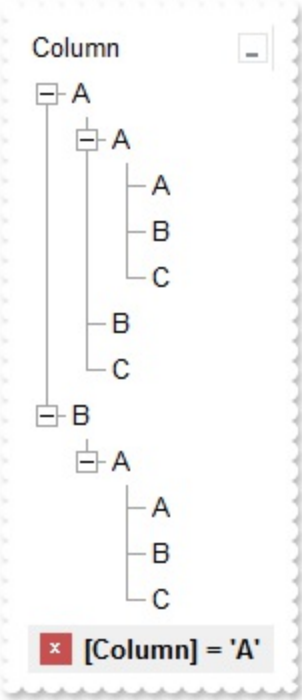
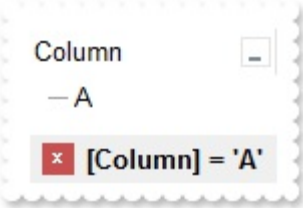
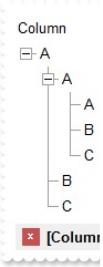
property Grid.FilterInclude as FilterIncludeEnum

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 Grid.FocusColumnIndex as Long

Specifies the index of focused column.

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

Use the FocusColumnIndex property to determine the focused column. Use the FocusItem property to determine the focused item. Use the TreeColumnIndex property to set the column that displays the hierarchy. Use the SearchColumnIndex property to set the index of the searching column. The SelectColumnInner property indicates the index of an inner cell that has the focus.

The control fires the FocusChanged event when the user changes:

- the focused item
- the focused column or an inner cell gets the focus.

property Grid.Font as IFontDisp

Retrieves or sets the control's font.

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

Use the Font object to change the control's font. Use the CellBold, CellItalic, CellUnderline, and CellStrikeOut properties to apply font attributes to cells. Use the ItemBold, ItemItalic, ItemUnderline, and ItemStrikeOut properties to apply font attributes to items. Use the HeaderBold, HeaderItalic, HeaderUnderline and HeaderStrikeOut properties to apply font attributes to the column header. 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 Grid1
   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_grid.GetFont();
font.SetName( "Tahoma" );
m_grid.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 AxGrid1
   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);
axGrid1.Font = font;
axGrid1.CtlRefresh();
```

The following VFP sample assigns by code a new font to the control:

```
with thisform.Grid1.Object
   .Font.Name = "Tahoma"
   .Refresh()
endwith
```

The following Template sample assigns by code a new font to the control:

```
Font
{
   Name = "Tahoma"
}
```

property Grid.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.

Use the ForeColor property to set the control's foreground color. If the control contains locked columns, (if the CountLockedColumns property is grater than 0, a locked column is a column non scrollable), use the ForeColorLock property to specify the foreground color for locked columns. Use the CellForeColor property to set the cell's foreground color. Use the ItemForeColor property to specify the item's foreground color. The control highlights the selected items only if the SelBackColor and BackColor properties have different values, and the SelForeColor and ForeColor properties have different values. Use the Def(exCellForeColor) property to change the foreground color for all cells in the column. The SelForeColor property is applied only if it is different that the control's foreground color.

The following VB sample sets the foreground color for the first column:

```
With Grid1.Columns(0)
   .Def(exCellForeColor) = RGB(255, 0, 0)
End With
```

The following C++ sample sets the foreground color for the first column:

```
#include "Column.h"
#include "Columns.h"
CColumns columns = m_grid.GetColumns();
CColumn column = columns.GetItem( COleVariant( long(0) ) );
column.SetDef(5, COleVariant( (long)RGB(255,0,0 ) ) );
```

The following VB.NET sample sets the foreground color for the first column:

```
With AxGrid1.Columns(0)
   .Def(EXGRIDLib.DefColumnEnum.exCellForeColor) = ToUInt32(Color.FromArgb(255, 0,
0))
End With
```

where the ToUInt32 function converts a Color expression to 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 sets the foreground color for the first column:

```
axGrid1.Columns[0].set_Def(EXGRIDLib.DefColumnEnum.exCellForeColor,
ToUInt32(Color.FromArgb(255, 0, 0)));
```

where the ToUInt32 function converts a Color expression to 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 VFP sample sets the foreground color for the first column:

```
with thisform.Grid1.Columns(0)
   .Def(5) = RGB(255,0,0)
endwith
```

property Grid.ForeColorHeader as Color

Specifies the header's foreground color.

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

Use the BackColorHeader and ForeColorHeader properties to define colors used to paint the control's header bar. Use the HeaderVisible property to show or hide the control's header. Use the HeaderHeight property to specify the height of the control's header bar. Use the LevelKey property to allow multiple levels header bar.

property Grid.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 control contains locked columns if the CountLockedColumn property is greater than zero (0). A locked column is fixed to the left side of the control, and it cannot be scrolled. If the CountLockedColumn property is greater than 0, the ForeColor property sets the foreground color for the unlocked area. The unlocked area is the area that contains scrollable columns. Use the Def(exCellForeColor) property to change the foreground color for all cells in the column.

property Grid.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 Grid.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 Grid.

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 Grid.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 Grid.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 Change event is fired anytime the cell's value is changed (CellValue property), 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:

```
' Change event - Occurs when the user changes the cell's content.
Private Sub Grid1_Change(ByVal Item As EXGRIDLibCtl.HITEM,ByVal ColIndex As
Long,NewValue As Variant)
   With Grid1
      Debug.Print( "Change event" )
   End With
End Sub

With Grid1
   .FreezeEvents True
   .BeginUpdate
   With .Columns
      .Add("C1").Def(exCellHasCheckBox) = True
      .Add "C2"
   End With
   With .Items
      .CellValue(.AddItem("SubItem 1.1"),1) = "SubItem 1.2"
      .CellValue(.AddItem("SubItem 2.1"),1) = "SubItem 2.2"
   End With
   .EndUpdate
   .FreezeEvents False
End With
```

property Grid.FullRowSelect as CellSelectEnum

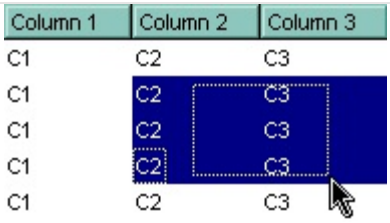
Enables full-row selection in the control.

Type	Description
CellSelectEnum	A CellSelectEnum expression that indicates whether the entire row is selected.

Use the FullRowSelect property to determine when the item or cell is selected. If the FullRowSelect property is exColumnSel, the SelectColumnIndex property determines the selected column. By default, the FullRowSelect property is exItemSel, and so the entire item is selected. If the FullRowSelect property is exRectSel property, the user can selects a range of cells by dragging. Use the Selected property to determine whether a cell is selected, if the FullRowSelect property is exRectSel. Use the SingleSel property to allow multiple items/cells in the selection. For instance, the FullRowSelect = True (boolean value) is the same as FullRowSelect = exItemSel, and FullRowSelect = False is the same as FullRowSelect = exColumnSel.

The following VB sample copies the selected cells to the clipboard, if the FullRowSelect property is exRectSel:

```
Private Sub Grid1_SelectionChanged()
  Dim strData As String
  With Grid1
    Dim i As Long, h As HITEM
    For i = 0 To .Items.SelectCount - 1
      h = .Items.SelectedItem(i)
      Dim c As Column
      For Each c In .Columns
        If (c.Selected) Then
          strData = strData + .Items.CellCaption(h, c.Index) + vbTab
        End If
      Next
      strData = strData + vbCrLf
    Next
  End With
  Clipboard.Clear
  Clipboard.SetText strData
End Sub
```



The following C++ sample copies the selected cells to the clipboard, if the FullRowSelect property is exRectSel:

```
#include "Column.h"
#include "Columns.h"
#include "Items.h"
void OnSelectionChangedGrid1()
{
   CString strData;
   CColumns cols = m_grid.GetColumns();
   CItems items = m_grid.GetItems();
   for ( long i = 0; i < items.GetSelectCount(); i++ )
   {
      COleVariant vtItem( items.GetSelectedItem( i ) );
      for ( long j = 0; j < cols.GetCount(); j++ )
      {
         COleVariant vtColumn( j );
         if ( cols.GetItem( vtColumn ).GetSelected() )
            strData += items.GetCellCaption(vtItem, vtColumn ) + "\t";
      }
      strData += "\r\n";
   }
   if ( OpenClipboard() )
   {
      EmptyClipboard();
      HGLOBAL hGlobal = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE,
strData.GetLength() );
      CopyMemory( GlobalLock( hGlobal ), strData.operator LPCTSTR(),
strData.GetLength() );
      GlobalUnlock( hGlobal );
      SetClipboardData( CF_TEXT, hGlobal );
      CloseClipboard();
   }
```

```
}
```

The following VB.NET sample copies the selected cells to the clipboard, if the FullRowSelect property is exRectSel:

```
Private Sub AxGrid1_SelectionChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles AxGrid1.SelectionChanged
   Dim strData As String = ""
   With AxGrid1
      Dim i As Integer, h As Integer, j As Integer
      For i = 0 To .Items.SelectCount - 1
         h = .Items.SelectedItem(i)
         For j = 0 To .Columns.Count - 1
            Dim c As EXGRIDLib.Column = .Columns(j)
            If (c.Selected) Then
               strData = strData + .Items.CellCaption(h, c.Index) + vbTab
            End If
         Next
         strData = strData + vbCrLf
      Next
   End With
   Clipboard.Clear()
   Clipboard.SetText(strData)
End Sub
```

The following C# sample copies the selected cells to the clipboard, if the FullRowSelect property is exRectSel:

```
private void axGrid1_SelectionChanged(object sender, System.EventArgs e)
{
   string strData = "";
   for (int i = 0; i < axGrid1.Items.SelectCount; i++)
   {
      for ( int j = 0; j < axGrid1.Columns.Count; j++ )
         if (axGrid1.Columns[j].Selected)
         {
            string cellData =
axGrid1.Items.get_CellCaption(axGrid1.Items.get_SelectedItem(i), j);
```

```
            strData += cellData + "\t";
        }
    strData += "\r\n";
    }
    Clipboard.Clear();
    Clipboard.SetText(strData);
}
```

The following VFP sample copies the selected cells to the clipboard, if the FullRowSelect property is exRectSel (SelectionChanged event):

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

with thisform.Grid1.Items
    local strData, i, j, cols
    strData = ""
    cols = thisform.Grid1.Columns
    for i = 0 to .SelectCount - 1
        .DefaultItem = .SelectedItem( i )
        for j = 0 to cols.Count - 1
            if ( cols.Item(j).Selected )
                strData = strData + .CellCaption(0,j) + chr(9)
            endif
        next
        strData = strData + chr(13) + chr(10)
    next
    _CLIPTEXT = strData
endwith
```

method Grid.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 values. The list includes the *collapsed* items, and the items are included as they are displayed (sorted, filtered). This option exports the values of cells. This option exports the values of the cells (CellValue 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 values of the cells (CellValue property) If missing, the Options parameter is 0. If the control displays no items, the result is an empty object (VT_EMPTY).

Return	Description
Variant	A safe array that holds the items in the control. If the control has a single column, the GetItems returns a single dimension array (object[]), else The safe array being returned has two dimensions (object[,]). The first

	dimension holds the collection of columns, and the second holds the cells.

The GetItems method to get a safe array that holds the items in the control. The GetItems method gets the items as they are displayed, sorted and filtered. Also, the GetItems method collect the child items as well, no matter if the parent item is collapsed. Use the PutItems method to load an array to the control. The method returns nothing if the control has no columns or items. Use the Items property to access the items collection. You can use the GetItems(1) method to get the list of handles for the items as they are displayed, sorted and filtered. The GetItems method returns an empty expression (VT_EMPTY), if there is no items in the result.

/NET Assembly:

The following C# sample converts the returned value to a object[] when the control contains a single column:

```
object[] Items = (object[])exgrid1.GetItems()
```

or when the control contains multiple columns, the syntax is as follows:

```
object[,] Items = (object[,])exgrid1.GetItems()
```

The following VB.NET sample converts the returned value to a Object() when the control contains a single column:

```
Dim Items As Object() = Exgrid1.GetItems()
```

or when the control contains multiple columns, the syntax is as follows:

```
Dim Items As Object(,) = Exgrid1.GetItems()
```

/COM version:

The following VB sample gets the items from a control and put them to the second one:

```
With Grid2
  .BeginUpdate
    .Columns.Clear
    Dim c As EXGRIDLibCtl.Column
    For Each c In Grid1.Columns
      .Columns.Add c.Caption
    Next
    .PutItems Grid1.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_grid2.BeginUpdate();
    CColumns columns = m_grid.GetColumns(), columns2 = m_grid2.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_grid.GetItems( vtMissing );
    m_grid2.PutItems( &vtItems, vtMissing );
m_grid2.EndUpdate();
```

The following C# sample gets the items from a control and put them to a second one:

```
axGrid2.BeginUpdate();
for (int i = 0; i < axGrid1.Columns.Count; i++)
    axGrid2.Columns.Add(axGrid1.Columns[i].Caption);
object vtItems = axGrid1.GetItems("");
axGrid2.PutItems(ref vtItems);
axGrid2.EndUpdate();
```

The following VB.NET sample gets the items from a control and put them to a second one:

```
With AxGrid2
    .BeginUpdate()
    Dim j As Integer
    For j = 0 To AxGrid1.Columns.Count - 1
        .Columns.Add(AxGrid1.Columns(j).Caption)
    Next
    Dim vtItems As Object
    vtItems = AxGrid1.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.Grid2
   .BeginUpdate()
   for i = 0 to thisform.Grid1.Columns.Count - 1
      .Columns.Add( thisform.Grid1.Columns(i).Caption )
   next
   local array vtItems[1]
   vtItems = thisform.Grid1.GetItems("")
   .PutItems( @vtItems )
   .EndUpdate()
endwith
```

property Grid.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. The GridLineStyle property to specify the style for horizontal or/and vertical gridlines in the control. 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 Grid.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.EXGRIDLib.GridLinesStyleEnum.exGridLinesHDash Or
exontrol.EXGRIDLib.GridLinesStyleEnum.exGridLinesVSolid
```

The following C# sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle = exontrol.EXGRIDLib.GridLinesStyleEnum.exGridLinesHDash |
exontrol.EXGRIDLib.GridLinesStyleEnum.exGridLinesVSolid;
```

The following Delphi sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle := Integer(EXGRIDLib.GridLinesStyleEnum.exGridLinesHDash) Or
Integer(EXGRIDLib.GridLinesStyleEnum.exGridLinesVSolid);
```

The following VFP sample shows dash style for horizontal gridlines, and solid style for vertical grid lines:

```
GridLineStyle = 36
```

method Grid.Group ()

Forces the control to do a regrouping of the columns.

Type	Description

The Group method forces the control to re-group the items. The AllowGroupBy property specifies whether the control supports Group-By feature. The Group method has no effect if the AllowGroupBy property is False. The Ungroup method un-groups the items in the control's list. During execution any of these methods, the IsGrouping property returns True. You can call the SortOrder property to sort and group by specified column. Use the SortType property to determine the way how the column is sorted. The AddGroupItem event is fired when a new grouping items is added to the control's list. *You can use the AddGroupItem event, to add headers or footers during grouping, customize the aggregate formula to be displayed on different columns, while dropping a column to the sortbar.* The Column.AllowGroupBy property may be used to prevent grouping a specific column. The AllowSort property indicates whether the user can sort a column by clicking the column's header.

method Grid.GroupUndoRedoActions (Count as Long)

Groups the next to current Undo/Redo Actions in a single block.

Type	Description
Count as Long	A Long expression that specifies the number of entries being grouped in a single block of actions, in the Undo/Redo queue.

A block may hold multiple Undo/Redo actions. Use the GroupUndoRedoActions method to group two or more entries in the Undo/Redo queue in a single block, so when a next Undo/Redo operation is performed, multiple actions may occur. For instance, moving several bars in the same time (multiple bars selection) is already recorded as a single block. Use the UndoRedoQueueLength property to specify the number of entries that Undo/Redo queue may store.

A block starts with StartBlock and ends with EndBlock when listed by UndoListAction/RedoListAction property as in the following sample:

```
StartBlock
RemoveItem;0
RemoveItem;1
RemoveItem;1
RemoveItem;1
EndBlock
```

property Grid.HasButtons as ExpandButtonEnum

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

Type	Description
ExpandButtonEnum	An ExpandButtonEnum expression that indicates whether the control displays a + button to the left of each parent item.

The HasButtons property has effect only if the data is displayed as a grid. Use the InsertItem property to let the control displays your data as a grid. Use the TreeColumnIndex property to select the column where the hierarchy is displayed. Use the LinesAtRoot property to let the control displays a line that links the root items of the control. Use the CellVAlignment property to specify where the +/- AND the cell's caption is displayed in the item's client area. For instance, you can't have the +/- sign aligned to the top of the cell, and its caption aligned to the bottom. The +/- signs are always centered to the cell's caption, only the cell's caption can be aligned to the top or to the bottom of the cell's client area. The HasButtonsCustom property specifies the index of icons being used for +/- signs on parent items, when HasButtons property is exCustom. In CardView mode, the HasButtons property specifies whether the control displays an expand/collapse button in the title of the card. The ImageSize property defines the size (width/height) of the expand/collapse glyphs.

The following VB sample changes the +/- button appearance:

```
With Grid1
   .HasButtons = ExpandButtonEnum.exWPlus
End With
```

The following C++ sample changes the +/- button appearance:

```
m_grid.SetHasButtons( 3 /*exWPlus*/ );
```

The following VB.NET sample changes the +/- button appearance:

```
With AxGrid1
   .HasButtons = EXGRIDLib.ExpandButtonEnum.exWPlus
End With
```

The following C# sample changes the +/- button appearance:

```
axGrid1.HasButtons = EXGRIDLib.ExpandButtonEnum.exWPlus;
```

The following VFP sample changes the +/- button appearance:

```
with thisform.Grid1
    .HasButtons = 3 && exWPlus
endwith
```

property Grid.HasButtonsCustom(Expanded as Boolean) as Long

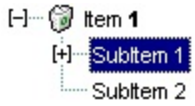
Specifies the index of icons for +/- signs when the HasButtons property is exCustom.

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.

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. The ImageSize property defines the size (width/height) of the expand/collapse glyphs.

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

```
With Grid1
  .BeginUpdate
    .Images
"gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExm

    .LinesAtRoot = exLinesAtRoot
    .HeaderVisible = False
    .HasButtons = exCustom
    .HasButtonsCustom(False) = 1
    .HasButtonsCustom(True) = 2
    .FullRowSelect = False
    .Columns.Add "Column 1"
    With .Items
      Dim h As HITEM
      h = .AddItem("Item 1")
      .InsertItem .InsertItem(h, , "SubItem 1.1"), , "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_grid.BeginUpdate();
m_grid.Images( COleVariant(
"gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExm
 ) );
m_grid.SetLinesAtRoot( -1 );
m_grid.SetHeaderVisible( FALSE );
m_grid.SetHasButtons( 4 /*exCustom*/ );
m_grid.SetHasButtonsCustom( FALSE, 1 );
m_grid.SetHasButtonsCustom( TRUE, 2 );
m_grid.GetColumns().Add( "Column 1" );
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
CItems items = m_grid.GetItems();
long h = items.AddItem( COleVariant( "Item 1" ) );
items.InsertItem( h, vtMissing, COleVariant( "SubItem 1" ) );
items.InsertItem( h, vtMissing, COleVariant( "SubItem 2" ) );
m_grid.EndUpdate();
```

The following VB.NET sample specifies different (as in the screen shot) +/- signs for the control:

```
With AxGrid1
   .BeginUpdate()

.Images("gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlkt

   .LinesAtRoot = EXGRIDLib.LinesAtRootEnum.exLinesAtRoot
   .HeaderVisible = False
   .HasButtons = EXGRIDLib.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:

```
axGrid1.BeginUpdate();
axGrid1.Images("gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk(

axGrid1.LinesAtRoot = EXGRIDLib.LinesAtRootEnum.exLinesAtRoot;
axGrid1.HeaderVisible = false;
axGrid1.HasButtons = EXGRIDLib.ExpandButtonEnum.exCustom;
axGrid1.set_HasButtonsCustom(false, 1);
axGrid1.set_HasButtonsCustom(true, 2);
axGrid1.Columns.Add("Column 1");
int h = axGrid1.Items.AddItem("Item 1");
axGrid1.Items.InsertItem(h, "", "SubItem 1");
axGrid1.Items.InsertItem(h, "", "SubItem 2");
axGrid1.EndUpdate();
```

The following VFP sample specifies different (as in the screen shot) +/- signs for the control:

```
with thisform.Grid1
    .BeginUpdate()
        local s
        s =
"gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExm

        s = s +
"1ntFptVrtltt1vuFxuVzul1u13vF5vV7vl9v1/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 Grid.HasLines as HierarchyLineEnum

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

Type	Description
HierarchyLineEnum	An HierarchyLinesEnum expression that indicates whether the control displays the hierarchy lines.

Use the HasLines property to hide the hierarchy lines. Use the LinesAtRoot property to allow control displays a line that links that root items of the control. Use the InsertItem method to insert new items to the control. Use HasButtons property to hide the buttons displayed at the left of each parent item. Use the DrawGridLines property to display grid lines. The GridLineStyle property to specify the style for horizontal or/and vertical gridlines in the control. Use the InsertControlItem property to insert an ActiveX item.

property Grid.HeaderAppearance as AppearanceEnum

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

Type	Description
AppearanceEnum	A boolean expression that specifies the appearance of the columns header.

Use the HeaderAppearance property to define the appearance of the columns header bar. The user can't resize the columns at runtime, if the HeaderAppearance property is None2. Use the ColumnsAllowSizing property to allow resizing the columns, when the control's header bar is not visible. Use the Appearance property to define the control's appearance. Use the HeaderVisible property to hide the control's header bar.

property Grid.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. Use the FormatLevel property to display multiple levels in the column's header. 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. *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 Grid1
    .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_grid.BeginUpdate();
m_grid.SetHeaderHeight( 32 );
```

```
m_grid.SetHeaderVisible( TRUE );
CColumn column1( V_DISPATCH( &m_grid.GetColumns().Add( "Column 1" ) ) );
   column1.SetHTMLCaption( "Line1<br>Line2" );
CColumn column2( V_DISPATCH( &m_grid.GetColumns().Add( "Column 2" ) ) );
   column2.SetHTMLCaption( "Line1<br>Line2" );
m_grid.EndUpdate();
```

The following VB.NET sample displays a header bar using multiple lines:

```
With AxGrid1
   .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:

```
axGrid1.BeginUpdate();
axGrid1.HeaderVisible = true;
axGrid1.HeaderHeight = 32;
EXGRIDLib.Column column1 = axGrid1.Columns.Add("Column 1") as EXGRIDLib.Column ;
column1.HTMLCaption = "Line1<br>Line2";
EXGRIDLib.Column column2 = axGrid1.Columns.Add("Column 2") as EXGRIDLib.Column;
column2.HTMLCaption = "Line1<br>Line2";
axGrid1.EndUpdate();
```

The following VFP sample displays a header bar using multiple lines:

```
with thisform.Grid1
   .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
```

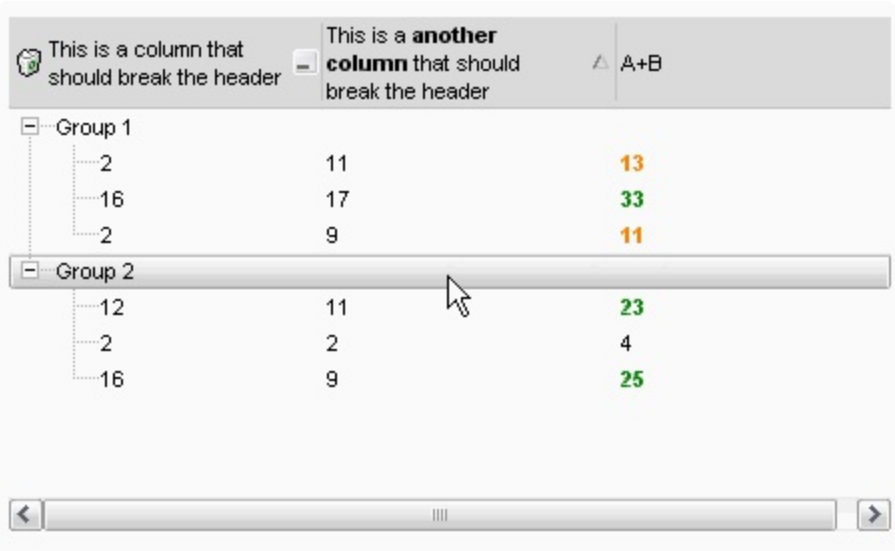
property Grid.HeaderSingleLine as Boolean

Specifies whether the control resizes the columns header and wraps the captions in single or multiple lines.

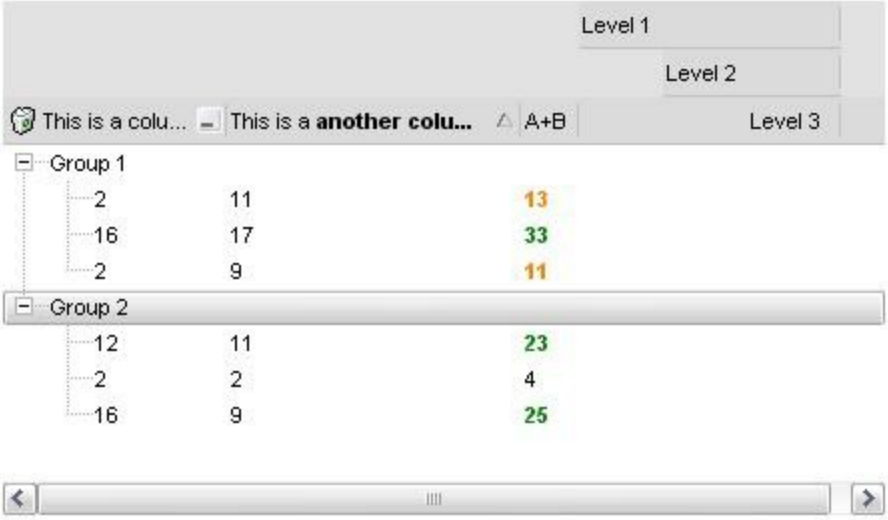
Type	Description
Boolean	A boolean expression that specifies whether the header displays single or multiple lines.

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.

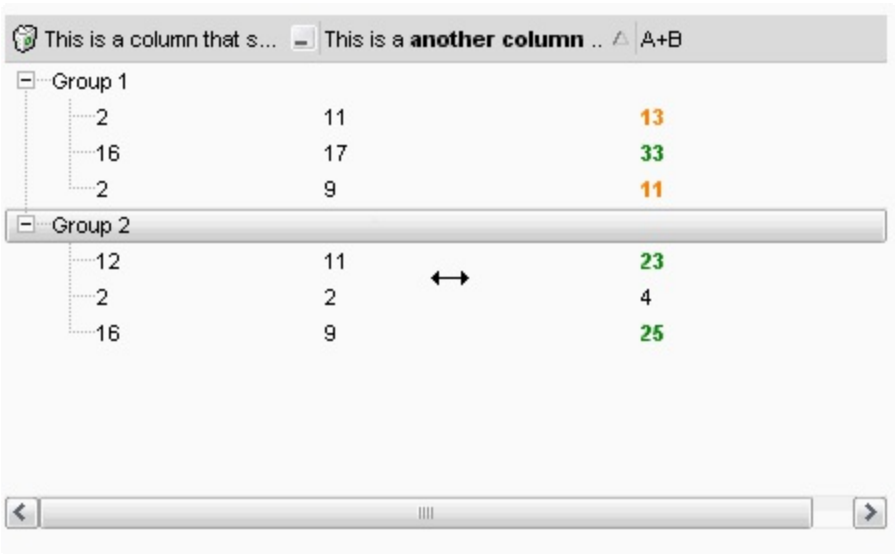
The following screen show shows the control's header while it displays a multiple lines (HeaderSingleLine = False):



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 Grid.HeaderVisible as Boolean

Retrieves or sets a value that indicates whether the the control's header is visible or hidden.

Type	Description
Boolean	A boolean expression that indicates whether the columns header bar is visible or hidden.

Use the HeaderVisible property to hide the columns header bar. Use the Visible property to hide a particular column. Use the ColumnFromPoint property to access the column from point. If the control's header bar is hidden, the ColumnFromPoint property returns -1. Use the LevelKey property to allow multiple levels header bar. Use the FormatLevel property to display multiple levels in the column's header. Use the HeaderHeight property to specify the height of the control's header bar. Use the BackColorHeader property to specify the header's background color. Use the AllowSizing property to disable resizing a column when user clicks the right margin of the column. Use the SortOnClick property to specify the action that control takes when the column's caption is clicked. Use the ColumnsAllowSizing property to allow resizing the columns, when the control's header bar is not visible. The Background(exCursorHoverColumn) property specifies the visual appearance of the column's header when the cursor hovers it.

property Grid.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, dialog box or control has the focus. Use the HideSelection property to hide the selected items when the control loses the focus. Use the SelBackColor property to indicate the background color for selected items. Use the SelForeColor property to specify the foreground color for selected items. Use the SelectItem property to select programmatically items. 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 Grid.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 Grid1
   .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 Grid1
   .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 Exgrid1
    .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 AxGrid1
    .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 'EXGRIDLib' for the library: 'ExGrid 1.0 Control Library'

   #import <ExGrid.dll>
   using namespace EXGRIDLib;
*/
EXGRIDLib::IGridPtr spGrid1 = GetDlgItem(IDC_GRID1)->GetControlUnknown();
spGrid1->BeginUpdate();
spGrid1->GetColumns()->Add(L"Def");
spGrid1->PutHotBackColor(RGB(0,0,128));
spGrid1->PutHotForeColor(RGB(255,255,255));
EXGRIDLib::IItemsPtr var_Items = spGrid1->GetItems();
   var_Items->AddItem("Item A");
   var_Items->AddItem("Item B");
   var_Items->AddItem("Item C");
spGrid1->EndUpdate();
```

C++ Builder

```
Grid1->BeginUpdate();
Grid1->Columns->Add(L"Def");
Grid1->HotBackColor = RGB(0,0,128);
Grid1->HotForeColor = RGB(255,255,255);
Exgridlib_tlb::IItemsPtr var_Items = Grid1->Items;
   var_Items->AddItem(TVariant("Item A"));
   var_Items->AddItem(TVariant("Item B"));
   var_Items->AddItem(TVariant("Item C"));
Grid1->EndUpdate();
```

C#

```
exgrid1.BeginUpdate();
exgrid1.Columns.Add("Def");
```

```
exgrid1.HotBackColor = Color.FromArgb(0,0,128);
exgrid1.HotForeColor = Color.FromArgb(255,255,255);
exontrol.EXGRIDLib.Items var_Items = exgrid1.Items;
   var_Items.AddItem("Item A");
   var_Items.AddItem("Item B");
   var_Items.AddItem("Item C");
exgrid1.EndUpdate();
```

JavaScript

```
<OBJECT classid="clsid:CD481F4D-2D25-4759-803F-752C568F53B7" id="Grid1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
   Grid1.BeginUpdate()

   Grid1.Columns.Add("Def")

   Grid1.HotBackColor = 8388608

   Grid1.HotForeColor = 16777215

   var var_Items = Grid1.Items

      var_Items.AddItem("Item A")

      var_Items.AddItem("Item B")

      var_Items.AddItem("Item C")

   Grid1.EndUpdate()

</SCRIPT>
```

C# for /COM

```
axGrid1.BeginUpdate();
axGrid1.Columns.Add("Def");
```

```
axGrid1.HotBackColor = Color.FromArgb(0,0,128);
axGrid1.HotForeColor = Color.FromArgb(255,255,255);
EXGRIDLib.Items var_Items = axGrid1.Items;
   var_Items.AddItem("Item A");
   var_Items.AddItem("Item B");
   var_Items.AddItem("Item C");
axGrid1.EndUpdate();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
   COM com_Items

   anytype var_Items




   super()


   exgrid1.BeginUpdate()

   exgrid1.Columns().Add("Def")

   exgrid1.HotBackColor(WinApi::RGB2int(0,0,128))

   exgrid1.HotForeColor(WinApi::RGB2int(255,255,255))

   var_Items = exgrid1.Items()
    com_Items = var_Items

      com_Items.AddItem("Item A")

      com_Items.AddItem("Item B")
```

```
        com_Items.AddItem("Item C")

    exgrid1.EndUpdate()

}
```

VFP

```
with thisform.Grid1
    .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 oGrid,var_Items

oGrid = form.Activex1.nativeObject
oGrid.BeginUpdate()
oGrid.Columns.Add("Def")
oGrid.HotBackColor = 0x800000
oGrid.HotForeColor = 0xffffff
var_Items = oGrid.Items
    var_Items.AddItem("Item A")
    var_Items.AddItem("Item B")
    var_Items.AddItem("Item C")
oGrid.EndUpdate()
```

XBasic (Alpha Five)

```
Dim oGrid as P
Dim var_Items as P

oGrid = topparent:CONTROL_ACTIVEX1.activex
oGrid.BeginUpdate()
oGrid.Columns.Add("Def")
oGrid.HotBackColor = 8388608
oGrid.HotForeColor = 16777215
var_Items = oGrid.Items
   var_Items.AddItem("Item A")
   var_Items.AddItem("Item B")
   var_Items.AddItem("Item C")
oGrid.EndUpdate()
```

Delphi 8 (.NET only)

```
with AxGrid1 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 Grid1 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 oGrid,var_Items

oGrid = ole_1.Object
oGrid.BeginUpdate()
oGrid.Columns.Add("Def")
oGrid.HotBackColor = RGB(0,0,128)
oGrid.HotForeColor = RGB(255,255,255)
var_Items = oGrid.Items
   var_Items.AddItem("Item A")
   var_Items.AddItem("Item B")
   var_Items.AddItem("Item C")
oGrid.EndUpdate()
```

property Grid.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 Grid1
   .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 Grid1
   .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 Exgrid1
    .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 AxGrid1
    .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 'EXGRIDLib' for the library: 'ExGrid 1.0 Control Library'

   #import <ExGrid.dll>
   using namespace EXGRIDLib;
*/
EXGRIDLib::IGridPtr spGrid1 = GetDlgItem(IDC_GRID1)->GetControlUnknown();
spGrid1->BeginUpdate();
spGrid1->GetColumns()->Add(L"Def");
spGrid1->PutHotBackColor(RGB(0,0,128));
spGrid1->PutHotForeColor(RGB(255,255,255));
EXGRIDLib::IItemsPtr var_Items = spGrid1->GetItems();
   var_Items->AddItem("Item A");
   var_Items->AddItem("Item B");
   var_Items->AddItem("Item C");
spGrid1->EndUpdate();
```

C++ Builder

```
Grid1->BeginUpdate();
Grid1->Columns->Add(L"Def");
Grid1->HotBackColor = RGB(0,0,128);
Grid1->HotForeColor = RGB(255,255,255);
Exgridlib_tlb::IItemsPtr var_Items = Grid1->Items;
   var_Items->AddItem(TVariant("Item A"));
   var_Items->AddItem(TVariant("Item B"));
   var_Items->AddItem(TVariant("Item C"));
Grid1->EndUpdate();
```

C#

```
exgrid1.BeginUpdate();
exgrid1.Columns.Add("Def");
exgrid1.HotBackColor = Color.FromArgb(0,0,128);
exgrid1.HotForeColor = Color.FromArgb(255,255,255);
exontrol.EXGRIDLib.Items var_Items = exgrid1.Items;
   var_Items.AddItem("Item A");
```

```
   var_Items.AddItem("Item B");
   var_Items.AddItem("Item C");
exgrid1.EndUpdate();
```

JavaScript

```
<OBJECT classid="clsid:CD481F4D-2D25-4759-803F-752C568F53B7" id="Grid1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
   Grid1.BeginUpdate()

   Grid1.Columns.Add("Def")

   Grid1.HotBackColor = 8388608

   Grid1.HotForeColor = 16777215

   var var_Items = Grid1.Items

      var_Items.AddItem("Item A")

      var_Items.AddItem("Item B")

      var_Items.AddItem("Item C")

   Grid1.EndUpdate()

</SCRIPT>
```

C# for /COM

```
axGrid1.BeginUpdate();
axGrid1.Columns.Add("Def");
axGrid1.HotBackColor = Color.FromArgb(0,0,128);
axGrid1.HotForeColor = Color.FromArgb(255,255,255);
EXGRIDLib.Items var_Items = axGrid1.Items;
   var_Items.AddItem("Item A");
```

```
    var_Items.AddItem("Item B");
    var_Items.AddItem("Item C");
axGrid1.EndUpdate();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Items

    anytype var_Items




    super()


    exgrid1.BeginUpdate()

    exgrid1.Columns().Add("Def")

    exgrid1.HotBackColor(WinApi::RGB2int(0,0,128))

    exgrid1.HotForeColor(WinApi::RGB2int(255,255,255))

    var_Items = exgrid1.Items()
     com_Items = var_Items

        com_Items.AddItem("Item A")

        com_Items.AddItem("Item B")

        com_Items.AddItem("Item C")

    exgrid1.EndUpdate()
```

```
}
```

VFP

```
with thisform.Grid1
   .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 oGrid,var_Items

oGrid = form.Activex1.nativeObject
oGrid.BeginUpdate()
oGrid.Columns.Add("Def")
oGrid.HotBackColor = 0x800000
oGrid.HotForeColor = 0xffffff
var_Items = oGrid.Items
   var_Items.AddItem("Item A")
   var_Items.AddItem("Item B")
   var_Items.AddItem("Item C")
oGrid.EndUpdate()
```

XBasic (Alpha Five)

```
Dim oGrid as P
Dim var_Items as P

oGrid = topparent:CONTROL_ACTIVEX1.activex
```

```
oGrid.BeginUpdate()
oGrid.Columns.Add("Def")
oGrid.HotBackColor = 8388608
oGrid.HotForeColor = 16777215
var_Items = oGrid.Items
   var_Items.AddItem("Item A")
   var_Items.AddItem("Item B")
   var_Items.AddItem("Item C")
oGrid.EndUpdate()
```

Delphi 8 (.NET only)

```
with AxGrid1 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 Grid1 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 oGrid,var_Items

oGrid = ole_1.Object
oGrid.BeginUpdate()
oGrid.Columns.Add("Def")
oGrid.HotBackColor = RGB(0,0,128)
oGrid.HotForeColor = RGB(255,255,255)
var_Items = oGrid.Items
  var_Items.AddItem("Item A")
  var_Items.AddItem("Item B")
  var_Items.AddItem("Item C")
oGrid.EndUpdate()
```

property Grid.HTMLPicture(Key as String) as Variant

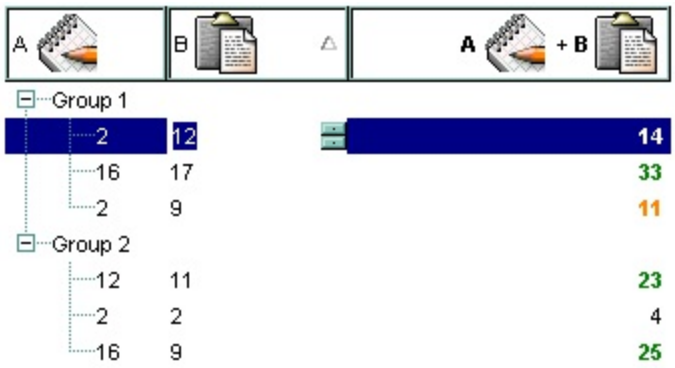
Adds or replaces a picture in HTML captions.

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")
   CellEditorVisible(h,0) = False
   CellEditorVisible(h,1) = False
   CellValueFormat(h,2) = 1
   h1 = InsertItem(h,,16)
   CellValue(h1,1) = 17
   h1 = InsertItem(h,,2)
   CellValue(h1,1) = 11
   h1 = InsertItem(h,,2)
   CellValue(h1,1) = 9
   ExpandItem(h) = True
   h = InsertItem(,,"Group 2")
```

```
    CellEditorVisible(h,0) = False
    CellEditorVisible(h,1) = False
    CellValueFormat(h,2) = 1
    h1 = InsertItem(h,,16)
    CellValue(h1,1) = 9
    h1 = InsertItem(h,,12)
    CellValue(h1,1) = 11
    h1 = InsertItem(h,,2)
    CellValue(h1,1) = 2
    ExpandItem(h) = True
    SelectItem(h) = True
}
EndUpdate
```

property Grid.hWnd as Long

Retrieves the control's window handle.

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

Use the hWnd property to get the handle of the control's window. Use the ItemWindowHost property to get the handle of the container window that host an ActiveX control. Use the Editing property to get the window's handle for the built-in editor that's visible and focused, while control is running in the edit mode. 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 Grid.HyperLinkColor as Color

Specifies the hyperlink color.

Type	Description
Color	A color expression that defines the color used by hyperlink cells.

The HyperLinColor property defines the cell's foreground color being used when cursor hovers a cell that has the CellHyperLink property is True. Use the CellForeColor property to change the cell's foreground color. Use the ItemForeColor property to change the item's foreground color. Use the ForeColor property to specify a color for the entire control. Use the SelForeColor property to specify the foreground color for selected items.

method Grid.Images (Handle as Variant)

Sets the control's image list at runtime.

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

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

The control provides an image list window, that's displayed at design time. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. Use the ShowImageList property to hide the image list window, at design time. At design time, the user can add new icons to the control's Images collection, by dragging icon files, exe files, etc, to the images list window. At runtime, the user can use the Images and ReplaceIcon method to change the Images collection. The Images collection is 1 based. Use the CellImage, CellImages properties to assign icons to a cell. Use the CellPicture property to load a picture in a cell.

The following VB sample adds two icons to the control's images collection using the BASE64 encoding strings:

```
With Grid1
   .BeginUpdate
   .Images
"gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExm

   .Columns.Add("Column 1").HeaderImage = 1
   .Items.CellImage(.Items.AddItem("Item 1"), 0) = 2
   .EndUpdate
End With
```

The following template adds two icons to the control's images collection using the BASE64 encoding strings:

```
BeginUpdate

Images("gBJJgBAICAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl

   Columns
   {
   Add("Column 1")
   {
      HeaderImage = 1
   }
```

```
    }
    Items
    {
    Dim h
    h = AddItem("Item 1")
    CellImage(h, 0) = 2
    }
EndUpdate
```

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

```
Grid1.Images ImageList1.hImageList
```

The control copies the images list, so destroying the source of the images list should not affect the images in the control.

The following C++ sample loads icons from a BASE64 encoded string:

```
#include "Items.h"
#include "Columns.h"
#include "Column.h"
m_grid.BeginUpdate();
CString s =
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn

s +=
"/xoAw9ZiFdxbAAGVxM5yOTzkPy+MzGRpmdx2kI2epGY1WgxmZl+Yyery2yyGHyeirGoo+(

s +=
"NbDDLO2xz5PlBi3O8x0EsZD7zuG8T1vrCD5uZE7zxM+CXQNB78RKw8RRbF8Rwyu0UPS/U

s +=
"kSSAAkqUU2nE20gmp5oo6JwH+eZ31EjJwB+eBn1K/AHnBWIfvwAZwACYAHsMy9cIMyFeE

m_grid.Images( COleVariant( s ) );
m_grid.GetColumns().Add( "Column 1" );
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
CItems items = m_grid.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_grid.EndUpdate();
```

The following C++ sample loads icon from a HIMAGELIST type:

```
SHFILEINFO sfi; ZeroMemory( &sfi, sizeof(sfi) );
HIMAGELIST hSysImageList = (HIMAGELIST)SHGetFileInfo(_T("C:\\"),  0, &sfi, sizeof
(SHFILEINFO), SHGFI_SMALLICON | SHGFI_SYSICONINDEX  );
m_grid.Images( _variant_t( (long)hSysImageList ) );
```

The following VB.NET sample loads icons from a BASE64 encoded string:

```
Dim s As String
With AxGrid1
   .BeginUpdate()
   s =
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn

   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:

```
axGrid1.BeginUpdate();
string s =
```

```
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn

s = s + "Poyf5xoojKAg";
axGrid1.Images(s);
axGrid1.Columns.Add("Column 1");
int h = axGrid1.Items.AddItem("Item 1");
axGrid1.Items.set_CellImage(h, 0, 1 );
h = axGrid1.Items.AddItem("Item 2");
axGrid1.Items.set_CellImages(h, 0,"2,3");
axGrid1.EndUpdate();
```

The following VFP sample loads icons from a BASE64 encoded string:

```
local s
With thisform.Grid1
   .BeginUpdate()
   s =
"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vmExn

   s = s +
"dr1fsFhsVjslls1ntFptVrtltt1vuFxuVzul1u13vF5vV7vl9v1/wGBnqAQEZwmCxFhYGLib/xoAw9

   s = s +
"Goo+03mM02Jzee029y2Ewum2+FnOTlGezHNx0b3/C3U258a4mP5HVvOw52s2fg2vH6ml

   s = s +
"kicAJnCbsNbDDLO2xz5PlBi3O8x0EsZD7zuG8T1vrCD5uZE7zxM+CXQNB78RKw8RRbF8Rwy

   s = s +
"wi/8iNPJMjSo0clvjMLuTHLkJTNCqVTSms2Tkq8jTzOcVP/BsePUocQLDQ9AJ3LtFUbR1Hqaiw

   s = s + "Poyf5xoojKAg"
   .Images(s)

   .Columns.Add("Column 1")
   With .Items
      .DefaultItem = .AddItem("Item 1")
```

```
        .CellImage(0, 0) = 1
        .DefaultItem = .AddItem("Item 2")
        .CellImages(0, 0) = "2,3"
    EndWith
    .EndUpdate()
EndWith
```

property Grid.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 Grid.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

By default, the Indent property is 22 pixels. 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 Grid.IsGrouping as Boolean

Indicates whether the control is grouping the items.

Type	Description
Boolean	A Boolean expression that specifies whether the control is grouping or ungrouping the items.

The IsGrouping property determines whether the control is grouping/ungrouping the items. The AllowGroupBy property specifies whether the control supports Group-By feature. For instance, during grouping, the control may expand or collapse items, you can use the IsGrouping property to determine if the BeforeExpandItem/AfterExpandItem events occur due user interaction or control's grouping operation. The GroupItem property indicates the index of the column being grouped for specified grouping item. The Group/Ungroup method groups or ungroup the control's list. During execution any of these methods, the IsGrouping property returns True. The LayoutChanged event is fired when the user changes the layout of the control, including dragging a column to the sort bar. The SortBarColumnsCount property indicates the number of the columns being grouped. The SortBarColumn property indicates the column being sorted giving its position in the sort bar.

property Grid.ItemFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS, ColIndex as Long, HitTestInfo as HitTestInfoEnum) as HITEM

Retrieves the item given a 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.
ColIndex as Long	A long expression that indicates on return the column where the point belongs.
HitTestInfo as HitTestInfoEnum	A HitTestInfoEnum expression that determines on return the position of the cursor within the cell.
HITEM	An 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.

The ItemFromPoint property returns:

- the **handle** of the item from the current cursor position, if the **X** and **Y** parameters are **-1**. The ItemFromPoint property returns 0, if not item is found.
- the **number** of rows from current cursor position to the last visible item, if the **X** is **0** and **Y** parameter is **-1**. The ItemFromPoint property returns 0, if the cursor hovers any item, else it returns a positive value, that indicate the number of items between the last visible item and the current cursor position. For instance, you can use this option, to add items to the cursor, once the user clicks the empty area of the items section of the control.

Use the ColumnFromPoint property to get the column from point (when the control's header is visible). Use the SelectableItem property to specify the user can select an item. The WordFromPoint property determines the word from the cursor.

The following VB sample prints the cell's value from the cursor:

```
Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
```

```
Single)
   ' Prints the cell over the cursor ( it doesn't include the inner cells )
   With Grid1
      Dim c As Long, hit As Long
      Dim h As HITEM
      h = .ItemFromPoint(-1, -1, c, hit)
      If Not (h = 0) Then
         Debug.Print .Items.CellCaption(h, c)
      End If
   End With
End Sub
```

The following VB sample prints the cell's value from the cursor (the sample doesn't print the inner cells that are created using the SplitCell property of the Items object) :

```
Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   ' Prints the cell over the cursor ( it doesn't include the inner cells )
   With Grid1
      Dim c As Long, hit As Long
      Dim h As HITEM
      h = .ItemFromPoint(-1, -1, c, hit)
      If Not (h = 0) Then
         Debug.Print .Items.CellValue(h, c)
      End If
   End With
End Sub
```

The following VB sample prints the cell's value from the cursor (the sample prints the caption of the inner cells too):

```
Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   ' Prints the cell over the cursor ( it includes the inner cells )
   With Grid1
      Dim c As Long, hit As Long
      Dim h As HITEM
      h = .ItemFromPoint(-1, -1, c, hit)
```

```
    If Not (h = 0) Or Not (c = 0) Then
        Debug.Print .Items.CellValue(h, c)
    End If
  End With
End Sub
```

The following VB sample displays the index of the icon being clicked, when the cell contains multiple icons:

```
Private Sub Grid1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As
Single)
  Dim i As HITEM, h As HitTestInfoEnum, c As Long
  With Grid1
    i = .ItemFromPoint(-1, -1, c, h)
  End With
  If Not (i = 0) Or Not (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 VB sample displays the cell's value from the cursor when hovering the inner controls (exgrid) too:

```
Private Function InsideItem(ByRef gObject As Object, ByRef c As Long, ByRef hit As
HitTestInfoEnum) As Long
  Dim i As Long
  i = gObject.ItemFromPoint(-1, -1, c, hit)
  If (i <> 0) Then
    If (IsEmpty(gObject.Items.ItemObject(i))) Then
      InsideItem = i
      Exit Function
    End If
    Set gObject = gObject.Items.ItemObject(i)
    With gObject
      i = gObject.ItemFromPoint(-1, -1, c, hit)
```

```
        InsideItem = i
    End With
  End If
End Function
```

And the MouseMove/ItemOLEEvent should look such as:

```
Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
  Dim c As Long, hit As HitTestInfoEnum, i As Long
  Dim g As Object
  Set g = Grid1.Object
  i = InsideItem(g, c, hit)
  Debug.Print g.Items.CellValue(i, c)
End Sub

Private Sub Grid1_ItemOleEvent(ByVal Item As EXGRIDLibCtl.HITEM, ByVal Ev As
EXGRIDLibCtl.IOleEvent)
  If (Ev.ID = -606) Then   ' Inside Mouse Move
    Dim c As Long, hit As HitTestInfoEnum, i As Long
    Dim g As Object
    Set g = Grid1.Object
    i = InsideItem(g, c, hit)
    Debug.Print g.Items.CellValue(i, c)
  End If
End Sub
```

The following C++ sample displays the cell's from the point:

```
void OnMouseMoveGrid1(short Button, short Shift, long X, long Y)
{
  long c = 0, hit = 0;
  long h = m_grid.GetItemFromPoint( -1, -1, &c, &h );
  if ( ( h != 0 ) || ( c != 0 ) )
  {
    COleVariant vtItem( h ), vtColumn( c );
    CItems items = m_grid.GetItems();
    CString strOutput;
```

```
        strOutput.Format( "Cell: %s\n" , items.GetCellCaption( vtItem, vtColumn ) );
        OutputDebugString( strOutput );
    }
}
```

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 OnMouseMoveGrid1(short Button, short Shift, long X, long Y)
{
    long c = 0, hit = 0, hItem = m_grid.GetItemFromPoint( -1, -1, &c, &hit );
    if ( ( hItem != 0 ) || ( c != 0 ) )
    {
        CItems items = m_grid.GetItems();
        COleVariant vtItem( hItem ), vtColumn( c );
        CString strCaption = V2S( &items.GetCellValue( 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 AxGrid1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_MouseMoveEvent) Handles AxGrid1.MouseMoveEvent
    With AxGrid1
        Dim i As Integer, c As Integer, hit As EXGRIDLib.HitTestInfoEnum
        i = .get_ItemFromPoint(-1, -1, c, hit)
        If (Not (i = 0) Or Not (c = 0)) Then
            Debug.WriteLine("Cell: " & .Items.CellValue(i, c) & " Hit: " & hit.ToString())
        End If
    End With
End Sub
```

The following C# sample displays the cell's from the point:

```
private void axGrid1_MouseMoveEvent(object sender,
AxEXGRIDLib._IGridEvents_MouseMoveEvent e)
{
    int c = 0;
    EXGRIDLib.HitTestInfoEnum hit;
    int i = axGrid1.get_ItemFromPoint( -1, -1, out c,out hit );
    if ( ( i != 0 ) || ( c != 0 ) )
    {
        object cap = axGrid1.Items.get_CellValue(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.Grid1
    .Items.DefaultItem = .ItemFromPoint( -1, -1, @c, @hit )
```

```
   if ( .Items.DefaultItem <> 0 ) or ( c <> 0 )
      wait window nowait .Items.CellValue( 0, c ) + " " + Str( hit )
   endif
endwith
```

property Grid.Items as Items

Retrieves the control's item collection.

Type	Description
Items	Defines the control' 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 Grid.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, before loading data to your control . The DefaultItemHeight property affects only items that are going to be added. It doesn't affect items already added. In CardView mode, uses the ViewModeOption(exCardViewVResizeLine) or ViewModeOption(exCardViewHResizeLine) property to show or hide the resizing lines. Also, the ItemsAllowSizing property allows resizing the cards at run-time.

property Grid.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/FocusColumnIndex property, indicates the focusing column, or 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 Grid.LinesAtRoot as LinesAtRootEnum

Link items at the root of the hierarchy.

Type	Description
LinesAtRootEnum	A LinesAtRootEnum expression that indicates whether the control links the items at the root of the hierarchy.

The control paints the hierarchy lines to the right if the Column's Alignment property is RightAlignment. The TreeColumnIndex property specifies the index of column where the hierarchy lines are painted. Use the Indent property to increase or decrease the amount, in pixels, that child items are indented relative to their parent items. Use the HasLines property to enhances the graphic representation of a tree control's hierarchy by drawing lines that link child items to their corresponding parent item. Use the InsertItem method to insert a child item. Use the InsertControlItem property to insert an ActiveX item.

method Grid.LoadXML (Source as Variant)

Loads an XML document from the specified location, using MSXML parser.

Type	Description
Source as Variant	An indicator of the object that specifies the source for the XML document. The object can represent a file name, a URL, an IStream, a SAFEARRAY, or an IXMLDOMDocument.

Return	Description
Boolean	A boolean expression that specifies whether the XML document is loaded without errors. If an error occurs, the method retrieves a description of the error occurred.

The LoadXML method uses the MSXML (MSXML.DOMDocument, XML DOM Document) parser to load XML documents, previously saved using the SaveXML method. The control is emptied when the LoadXML method is called, and so the columns and items collection are emptied before loading the XML document. The LoadXML method adds a new column for each **<column>** tag found in the **<columns>** collection. Properties like Caption, HTMLCaption, Image, Visible, LevelKey, DisplayFilterButton, DisplayFilterPattern, FilterType, Width and Position are fetched for each column found in the XML document. The control fires the AddColumn event for each found column. The **<items>** xml element contains a collection of **<item>** objects. Each <item> object holds information about an item in the control, including its cells, or child items. Each item contains a collection of **<cell>** objects that defines the cell for each column. The Expanded attribute specifies whether an item is expanded or collapsed, and it carries the value of the ExpandItem property.

The XML format looks like follows:

```
- <Content Author Component Version ...>
  - <Columns>
      <Column Caption Position Width HTMLCaption LevelKey DisplayFilterButton
DisplayFilterPatter FilterType ... />
      <Column Caption Position Width HTMLCaption LevelKey DisplayFilterButton
DisplayFilterPatter FilterType ... />
      ...
    </Columns>
  - <Items>
    - <Item Expanded ...>
        <Cell Value ValueFormat Images Image ... />
        <Cell Value ValueFormat Images Image ... />
```

```
        ...
      - <Items>
        - <Item Expanded ...>
        - <Item Expanded ...>
        ....
        </Items>
      </Item>
    </Items>
  </Content>
```

property Grid.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 marks the searching column by drawing a rectangle around it. The SearchColumnIndex property determines the index of the searching column. Use the MarkSearchColumn property to hide the searching column. By default, the MarkSearchColumn property is True. 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.

property Grid.MarkTooltipCells as Boolean

Retrieves or sets a value that indicates whether the control marks the cells that have tool tips.

Type	Description
Boolean	A boolean expression that indicates whether the control marks the cells that have tool tips.

By default, the MarkTooltipCells property is False. If the MarkTooltipCells property is True, the control paints on the right side of the cell a sign that indicates that the cell has associated a tool tip. Use the CellTooltip property to associate a tool tip to a cell. Use the TooltipCellsColor property to change the color used to sign cells that have tool tips. Use the MarkTooltipCellsImage property to assign a different look for signs to mark the cells that have tooltips.

property Grid.MarkTooltipCellsImage as Long

Specifies a value that indicates the index of icon being displayed in the cells that have tooltips.

Type	Description
Long	A long expression that indicates the index of icon being displayed when a cell has tooltip assigned.

By default, the MarkTooltipCellsImage property is 0. The MarkTooltipCellsImage property has effect only if the MarkTooltipCells property is True. Use the Images method to assign new icons to the control. By default, if the control can't find the icon specified by the MarkTooltipCellsImage property, it paints the default sign to mark the cells that have the tooltips.

The following screen shot shows how the control marks by default, the cells that have tooltip:

The following screen shot shows how the control marks the cells that have tooltip, once that MarkTooltipCellsImage property points to an icon in the Images collection:

method Grid.OLEDrag ()

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

Type	Description

The method is only for internal use.

property Grid.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. 0 means no drag and drop support, 1 means manual support.

In the /NET Assembly, you have to use the AllowDrop property as explained here:

- *https://www.exontrol.com/sg.jsp?content=support/faq/net/#dragdrop*

By default, the OLEDropMode property is exOLEDropNone. Curently, the control supports only manual OLE Drag and Drop operation. Use the SelectByDrag property to disable selecting multiple items by dragging. The SingleSel property controls the number of items that the user may select. For instance, if the SingleSel property is True, the user can't select multiple items, and so a single item may be selected at the time. If the SingleSel property is False, the user can select multiple items using the mouse, keyboard or both.

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.

See the OLEStartDrag and OLEDragDrop events for more details about implementing drag and drop operations into the ExGrid control.

property Grid.Picture as IPictureDisp

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

Type	Description
IPictureDisp	A Picture object that indicates the control's picture.

Use the Picture property to load a picture on the control's background. Use the PictureDisplay property to arrange the picture on the control's background. Use the SelBackMode property to define how the selected items are painted. Use the PictureLevelHeader property to display a picture on the control's header bar when it displays the columns using multiple levels. Use the CellPicture property to assign a picture to a cell. Use the BackColor property to specify the control's background color.

property Grid.PictureDisplay as PictureDisplayEnum

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

Type	Description
PictureDisplayEnum	A PictureDisplayEnum expression that indicates the way how the control's picture is displayed.

Use the Picture property to load a picture into the control's background. Use the PictureDisplay property to arrange how the control's picture is displayed on its background. Use the SelBackMode property to define how the selected items are painted. Use the CellPicture property to assign a picture to a cell. Use the BackColor property to specify the control's background color.

property Grid.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.

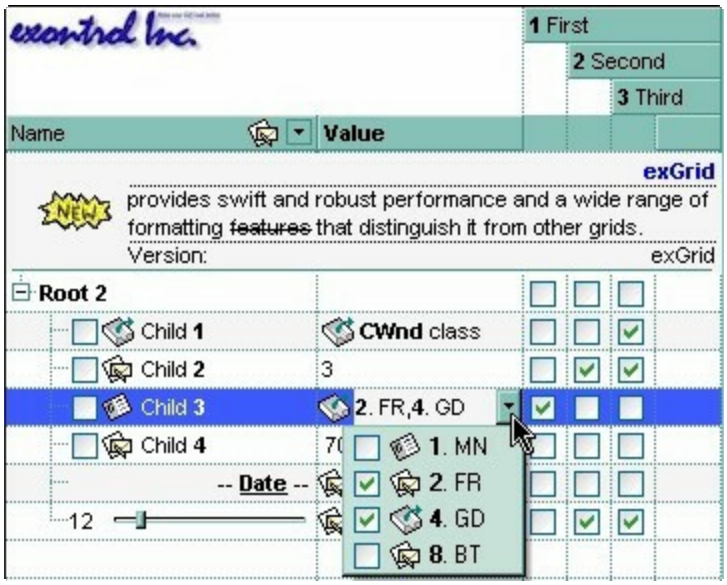
Use the PictureDisplayLevelHeader property to arrange the picture on the control's multiple levels header bar. Use the PictureLevelHeader property to load a picture on the control's header bar when it displays multiple levels. The control's header bar displays multiple levels if there are two or more neighbor columns with the same non empty level key. Use the LevelKey property to specify the control's level key.

property Grid.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 Grid.PutItems (Items as Variant, [Parent as Variant])

Adds an array of integer, long, date, string, double, float, or variant arrays to the Grid, beginning at Index.

Type	Description
Items as Variant	An array that control uses to fill with. The array can be one or two- dimensional. If the array is one-dimensional, the control requires one column being added before calling the PutItems method. If the Items parameter indicates a two-dimensional array, the first dimension defines the columns, while the second defines the number of items to be loaded. For instance, a(2,100) means 2 columns and 100 items.
Parent as Variant	A long expression that specifies the handle of the item where the array is being inserted, or 0 if missing

The PutItems method loads items from a safe array. The Parent parameter of the PutItems method specifies the handle of the item where the array is being inserted as child items. Use the 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.

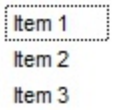
The following VB 6 sample loads a flat array to a single column control (and shows as in the following picture):

```
With Grid1
  .BeginUpdate
  .Columns.Add "Column 1"
  .PutItems Array("Item 1", "Item 2", "Item 3")
  .EndUpdate
End With
```

or similar for /NET Assembly version:

```
With Exgrid1
  .BeginUpdate()
  .Columns.Add("Column 1")
  .PutItems(New String() {"Item 1", "Item 2", "Item 3"})
```

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

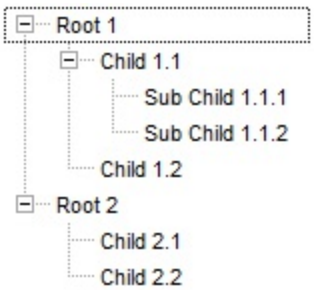


The following VB 6 sample loads a hierarchy to a single column control (and shows as in the following picture):

```
With Grid1
    .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 Exgrid1
    .BeginUpdate()
    .LinesAtRoot = exontrol.EXGRIDLib.LinesAtRootEnum.exLinesAtRoot
    .Columns.Add("")
    .PutItems(New Object() {"Root 1", New Object() {"Child 1.1", New String() {"Sub Child
1.1.1", "Sub Child 1.1.2"}, "Child 1.2"}, "Root 2", New String() {"Child 2.1", "Child 2.2"}})
    .EndUpdate()
End With
```



The following VB 6 sample loads a list of items, in a three columns control (as shown in the following picture):

```
Dim v(2, 2) As String
```

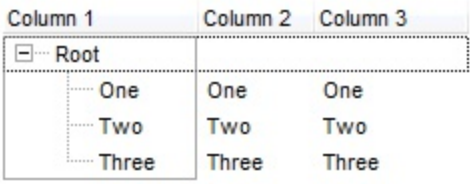
```
v(0, 0) = "One"
v(0, 1) = "Two"
v(0, 2) = "Three"
v(1, 0) = "One"
v(1, 1) = "Two"
v(1, 2) = "Three"
v(2, 0) = "One"
v(2, 1) = "Two"
v(2, 2) = "Three"

With Grid1
   .BeginUpdate
   .Columns.Add "Column 1"
   .Columns.Add "Column 2"
   .Columns.Add "Column 3"

   .PutItems v
   .EndUpdate
End With
```

Column 1	Column 2	Column 3
One	One	One
Two	Two	Two
Three	Three	Three

The following VB 6 sample loads a list of items, in a three columns control (as shown in the following picture):

```
Dim v(2, 2) As String
v(0, 0) = "One"
v(0, 1) = "Two"
v(0, 2) = "Three"
v(1, 0) = "One"
v(1, 1) = "Two"
v(1, 2) = "Three"
v(2, 0) = "One"
v(2, 1) = "Two"
v(2, 2) = "Three"
```

```
With Grid1
   .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 Grid1
   .BeginUpdate

      .ColumnAutoResize = False
      .MarkSearchColumn = False
      .DrawGridLines = True
      ' Adds a column for eac 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()

        'Changes the editor of the "Photo" column
        .Columns("Photo").Editor.EditType = PictureType
    .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_grid.BeginUpdate();
            m_grid.SetColumnAutoResize( FALSE );
            CColumns columns = m_grid.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_grid.PutItems( &spRecordset->GetRows(-1), vtMissing );
            m_grid.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 Grid.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 state being changed. True means checked, False means un-checked.
Long	A long expression that indicates the index of icon used for cells of radio type. The last 7 bits in the high significant byte of the long expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. If the index is not valid the default icon is used.

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 following VB sample changes the radio buttons appearance (the control's icons list should be loaded before):

```
With Grid1
  .BeginUpdate
    .Columns.Add "Radio"
    For i = 0 To 2
      Dim h As HITEM
      h = .Items.AddItem("Option " & i)
      .Items.CellHasRadioButton(h) = True
      .Items.CellRadioGroup(h) = 1234
    Next

    .RadioImage(True) = 1        ' Changes the icon for checked radio cells.
```

```
    .RadioImage(False) = 2        ' Changes the icon for un-checked radio cells.
  .EndUpdate
End With
```

The following VB sample changes the default icon for the cells of radio type:

```
Grid1.RadioImage(True) = 1          ' Sets the icon for cells of radio type that are checked
Grid1.RadioImage(False) = 2         ' Sets the icon for cells of radio type that are
unchecked
```

The Grid1.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_grid.SetRadioImage( TRUE, 1 );
m_grid.SetRadioImage( FALSE, 2 );
```

The following VB.NET sample changes the default icon for the cells of radio type:

```
With AxGrid1
  .set_RadioImage(True, 1)
  .set_RadioImage(False, 2)
End With
```

The following C# sample changes the default icon for the cells of radio type:

```
axGrid1.set_RadioImage(true, 1);
axGrid1.set_RadioImage(false, 2);
```

The following VFP sample changes the default icon for the cells of radio type:

```
with thisform.Grid1
  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 Grid.RClickSelect as Boolean

Retrieves or sets a value that indicates whether an item is selected using right mouse button.

Type	Description
Boolean	A boolean expression that indicates whether an item is selected using right mouse button.

Use the RClickSelect property to allow users select items using the right click. By default, the RClickSelect property is False. The control fires the SelectionChanged event when user selects an item. Use the SelectItem property to select programmatically select an item. Use the SelectCount property to get the number of selected items. Use the SelectedItem property to get the selected item. Use the FocusItem property to get the focused item. Use the ItemFromPoint property to retrieve an item from the point.

property Grid.ReadOnly as ReadOnlyEnum

Retrieves or sets a value that indicates whether the control is read only.

Type	Description
ReadOnlyEnum	A ReadOnlyEnum expression that indicates whether the control is read only.

The ReadOnly property makes the control read only. Use the Enabled property to disable the control. Use the Locked property to lock an editor. If the control is read only, the Edit or Change event is not fired. Use the CellEditorVisible property to hide the cell's editor. Use the SelectableItem property to specify the user can select an item.

method Grid.Redo ()

Redoes the next action in the control's Redo queue.

Type	Description

Call the Redo method to Redo the last control operation. The Redo method have effect only if the AllowUndoRedo property is True. The CTRL+Y redoes the next action in the control's Redo queue, while the CTRL+Z performs the last undo operation. Call the Undo method to Undo the last control operation. The CanUndo property retrieves a value that indicates whether the control may perform the last Undo operation. The CanRedo property retrieves a value that specifies whether the control can execute the next operation in the control's Redo queue. The URChange(exUndo/exRedo) event notifies your application whenever an Undo/Redo operation is performed.

The records of the Undo/Redo queue may contain actions in the following format:

- "**AddItem**;ITEMINDEX", indicates that a new item has been created
- "**RemoveItem**;ITEMINDEX", indicates that an item has been removed
- "**ChangeItemPos**;ITEMINDEX", indicates that an item changes its position or / and parent
- "**ChangeCellValue**;ITEMINDEX;CELLINDEX", indicates that the cell's value has been changed
- "**ChangeCellState**;ITEMINDEX;CELLINDEX", indicates that the cell's state has been changed

Also, the Undo/Redo queue may include:

- "**StartBlock**", specifies that a block of operations begins
- "**EndBlock**", specifies that a block of operations ends

The RedoListAction property lists the Redo actions that can be performed in the control. Use the RedoRemoveAction method to remove the first action from the redo queue.

property Grid.RedoListAction ([Action as Variant], [Count as Variant]) as String

Lists the Redo actions that can be performed in the control.

Type	Description
Action as Variant	[optional] A long expression that specifies the action being listed. If missing or -1, all actions are listed. The Action parameter can be one of the following: • exListUndoRedoAddItem(13) ~ "**AddItem**;ITEMINDEX", indicates that a new item has been created • exListUndoRedoRemoveItem(14) ~ "**RemoveItem**;ITEMINDEX", indicates that an item has been removed • exListUndoRedoChangeItemPos(15) ~ "**ChangeItemPos**;ITEMINDEX", indicates that an item changes its position or / and parent • exListUndoRedoChangeCellValue(16) ~ "**ChangeCellValue**;ITEMINDEX;CELLINDEX", indicates that the cell's value has been changed • exListUndoRedoChangeCellState(17) ~ "**ChangeCellState**;ITEMINDEX;CELLINDEX", indicates that the cell's state has been changed For instance, RedoListAction(13) shows only AddItem actions in the redo stack.
Count as Variant	[optional] A long expression that indicates the number of actions being listed. If missing or -1, all actions are listed. For instance, RedoListAction(13,1) shows only the first AddItem action being added to the redo stack.
String	A String expression that lists the Redo actions that may be performed.

The RedoListAction property lists the Redo actions that can be performed in the control. The URChange(exUndo/exRedo) event notifies your application whenever an Undo/Redo operation is performed. The UndoListAction property lists the actions that the user may perform by doing Undo operations. The CanRedo property specifies whether a redo operation can be performed if CTRL+Y key is pressed. Use the RedoRemoveAction method to remove the first action from the redo queue.

The records of the Undo/Redo queue may contain actions in the following format:

- "**AddItem**;ITEMINDEX", indicates that a new item has been created
- "**RemoveItem**;ITEMINDEX", indicates that an item has been removed
- "**ChangeItemPos**;ITEMINDEX", indicates that an item changes its position or / and parent
- "**ChangeCellValue**;ITEMINDEX;CELLINDEX", indicates that the cell's value has been changed
- "**ChangeCellState**;ITEMINDEX;CELLINDEX", indicates that the cell's state has been changed

Also, the Undo/Redo queue may include:

- "**StartBlock**", specifies that a block of operations begins
- "**EndBlock**", specifies that a block of operations ends

Each action is on a single line, and each field is separated by ; character. The lines are separated by "\r\n" characters (vbCrLf in VB).

Here's a sample format of the RedoListAction property may get:

```
AddItem;0
AddItem;1
AddItem;2
ChangeCellState;2;0
AddItem;3
ChangeCellState;3;0
StartBlock
RemoveItem;0
RemoveItem;1
RemoveItem;1
RemoveItem;1
EndBlock
```

The following VB sample splits the RedoListAction value and adds each action to a listbox control:

```
List1.Clear
Dim s() As String
s = Split(Grid1.RedoListAction, vbCrLf)
For i = LBound(s) To UBound(s)
```

```
        List1.AddItem s(i)
Next
```

method Grid.RedoRemoveAction ([Action as Variant], [Count as Variant])

Removes the last the redo actions that can be performed in the control.

Type	Description
Action as Variant	[optional] A long expression that specifies the action being removed. If missing or -1, all actions are removed. The Action parameter can be one of the following: • exListUndoRedoAddItem(13) ~ "**AddItem**;ITEMINDEX", indicates that a new item has been created • exListUndoRedoRemoveItem(14) ~ "**RemoveItem**;ITEMINDEX", indicates that an item has been removed • exListUndoRedoChangeItemPos(15) ~ "**ChangeItemPos**;ITEMINDEX", indicates that an item changes its position or / and parent • exListUndoRedoChangeCellValue(16) ~ "**ChangeCellValue**;ITEMINDEX;CELLINDEX", indicates that the cell's value has been changed • exListUndoRedoChangeCellState(17) ~ "**ChangeCellState**;ITEMINDEX;CELLINDEX", indicates that the cell's state has been changed For instance, RedoListAction(13) removes only AddItem actions in the redo stack.
Count as Variant	[optional] A long expression that indicates the number of actions to be removed. If missing or -1, all actions are removed. For instance, RedoListAction(13,1) removes only the first AddItem action from the redo stack.

Use the RedoRemoveAction method to remove the first action from the redo queue. Use the RedoRemoveAction() (with no parameters) to remove all redo actions. The RedoListAction property retrieves the list of actions that an redo operation can perform. The UndoRemoveAction method removes the last action to be performed if the Undo method is invoked.

The records of the Undo/Redo queue may contain actions in the following format:

- "**AddItem**;ITEMINDEX", indicates that a new item has been created
- "**RemoveItem**;ITEMINDEX", indicates that an item has been removed
- "**ChangeItemPos**;ITEMINDEX", indicates that an item changes its position or / and parent
- "**ChangeCellValue**;ITEMINDEX;CELLINDEX", indicates that the cell's value has been changed
- "**ChangeCellState**;ITEMINDEX;CELLINDEX", indicates that the cell's state has been changed

Also, the Undo/Redo queue may include:

- "**StartBlock**", specifies that a block of operations begins
- "**EndBlock**", specifies that a block of operations ends

.

method Grid.Refresh ()

Refreshes the control's content.

Type	Description

Use the Refresh method whenever you need to refresh the control. Use the BeginUpdate and EndUpdate method ay time when the control requires more changes at one time. Use the hWnd property to get the handle of the control's window.

The following VB sample calls the Refresh method:

```
Grid1.Refresh
```

The following C++ sample calls the Refresh method:

```
m_grid.Refresh();
```

The following VB.NET sample calls the Refresh method:

```
AxGrid1.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:

```
axGrid1.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.Grid1.Object.Refresh()
```

method Grid.RemoveSelection ()

Removes the selected items (including the descendents)

Type	Description

The RemoveSelection method removes the selected items (including the descendents). The RemoveItem method removes a specific item (if it has not child items). The UnselectAll method unselects all items in the list.

method Grid.ReplaceIcon ([Icon as Variant], [Index as Variant])

Adds a new icon, replaces an icon or clears the control's image list.

Type	Description
Icon as Variant	A long expression that indicates the icon's handle. By default, the Icon parameter is 0, if it is missing.
Index as Variant	A long expression that indicates the index where icon is inserted. By default, the Index parameter is -1, if it is missing.

Return	Description
Long	A long expression that indicates the index of the icon in the images collection

Use the ReplaceIcon property to add, remove or replace an icon in the control's images collection. Also, the ReplaceIcon property can clear the images collection. Use the Images method to attach an image list to the control. The ImageSize property defines the size (width/height) of the icons within the control's Images collection.

The following sample shows how to add a new icon to control's images list:

i = Grid1.ReplaceIcon(LoadPicture("d:\icons\help.ico").Handle), in this case the i specifies the index where the icon was added

The following sample shows how to replace an icon into control's images list::

i = Grid1.ReplaceIcon(LoadPicture("d:\icons\help.ico").Handle, 0), in this case the i is zero, because the first icon was replaced.

The following sample shows how to remove an icon from control's images list:

Grid1.ReplaceIcon 0, i, in this case the i must be the index of the icon that follows to be removed

The following sample shows how to clear the control's icons collection:

Grid1.ReplaceIcon 0, -1

property Grid.RightToLeft as Boolean

Indicates whether the component should draw right-to-left for RTL languages.

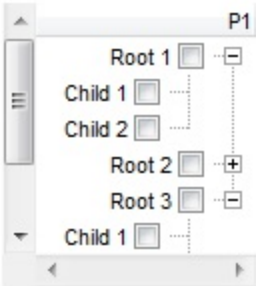
Type	Description
Boolean	A boolean expression that specifies whether the control is drawn from right to left or from left to right.

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:

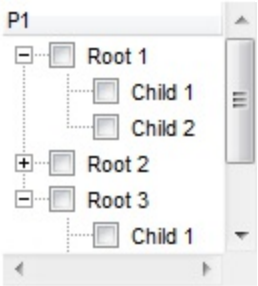


(By default) Changing the RightToLeft property on False does the following:

- 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 Grid1
   .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 AxGrid1
   .BeginUpdate
   .ScrollBars = EXGRIDLib.ScrollBarsEnum.exDisableBoth
   .LinesAtRoot = EXGRIDLib.LinesAtRootEnum.exLinesAtRoot
```

```
    With .Columns.Add("P1")
        .Def(EXGRIDLib.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 'EXGRIDLib' for the library: 'ExGrid 1.0 Control Library'

    #import <ExGrid.dll>
    using namespace EXGRIDLib;
*/
EXGRIDLib::IGridPtr spGrid1 = GetDlgItem(IDC_GRID1)->GetControlUnknown();
spGrid1->BeginUpdate();
spGrid1->PutScrollBars(EXGRIDLib::exDisableBoth);
spGrid1->PutLinesAtRoot(EXGRIDLib::exLinesAtRoot);
EXGRIDLib::IColumnPtr var_Column = ((EXGRIDLib::IColumnPtr)(spGrid1->GetColumns()-
>Add(L"P1")));
    var_Column->PutDef(EXGRIDLib::exCellHasCheckBox,VARIANT_TRUE);
    var_Column->PutPartialCheck(VARIANT_TRUE);
EXGRIDLib::IItemsPtr var_Items = spGrid1->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);
spGrid1->PutRightToLeft(VARIANT_TRUE);
```

```
spGrid1->EndUpdate();
```

The following C# sample shows how to change the order of the columns from right to left

```
axGrid1.BeginUpdate();
axGrid1.ScrollBars = EXGRIDLib.ScrollBarsEnum.exDisableBoth;
axGrid1.LinesAtRoot = EXGRIDLib.LinesAtRootEnum.exLinesAtRoot;
EXGRIDLib.Column var_Column = (axGrid1.Columns.Add("P1") as EXGRIDLib.Column);
   var_Column.set_Def(EXGRIDLib.DefColumnEnum.exCellHasCheckBox,true);
   var_Column.PartialCheck = true;
EXGRIDLib.Items var_Items = axGrid1.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);
axGrid1.RightToLeft = true;
axGrid1.EndUpdate();
```

The following VFP sample shows how to change the order of the columns from right to left

```
with thisform.Grid1
   .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 AxGrid1 do
begin
   BeginUpdate();
   ScrollBars := EXGRIDLib.ScrollBarsEnum.exDisableBoth;
   LinesAtRoot := EXGRIDLib.LinesAtRootEnum.exLinesAtRoot;
   with (Columns.Add('P1') as EXGRIDLib.Column) do
   begin
      Def[EXGRIDLib.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 Grid.SaveXML (Destination as Variant)

Saves the control's content as XML document to the specified location, using the MSXML parser.

Type	Description

Destination as Variant	This object can represent a file name, an XML document object, or a custom object that supports persistence as follows:
	
	• String - Specifies the file name. Note that this must be a file name, rather than a URL. The file is created if necessary and the contents are entirely replaced with the contents of the saved document. For example:
	
	`Grid1.SaveXML("sample.xml")`
	
	• XML Document Object. For example:
	
	`Dim xmldoc as Object`
	`Set xmldoc = CreateObject("MSXML.DOMDocument")`
	`Grid1.SaveXML(xmldoc)`
	
	• Custom object supporting persistence - Any other custom COM object that supports **QueryInterface** for **IStream**, **IPersistStream**, or **IPersistStreamInit** can also be provided here and the document will be saved accordingly. In the **IStream** case, the **IStream::Write** method will be called as it saves the document; in the **IPersistStream** case, **IPersistStream::Load** will be called with an **IStream** that supports the **Read**, **Seek**, and **Stat** methods.

Return	Description
Boolean	A Boolen expression that specifies whether saving the XML document was ok.

The SaveXML method uses the MSXML (MSXML.DOMDocument, XML DOM Document) parser to save the control's data in XML documents. The LoadXML method loads XML documents being created with SaveXML method. The SaveXML method saves each column in **<column>** elements under the **<columns>** collection. Properties like Caption, HTMLCaption, Image, Visible, LevelKey, DisplayFilterButton, DisplayFilterPattern, FilterType, Width and Position are saved for each column in the control. The **<items>** xml element saves a collection of **<item>** objects. Each <item> object holds information about

an item in the control, including its cells or child items. Each item saves a collection of **<cell>** objects that defines the cell for each column. The Expanded attribute specifies whether an item is expanded or collapsed, and it carries the value of the ExpandItem property.

The control saves the control's data in XML format like follows:

```
- <Content Author Component Version ...>
  - <Chart FirstVisibleDate ...>
    - <Levels>
        <Level Label Unit Count />
        <Level Label Unit Count />
        ...
      </Levels>
    - <Links>
        <Link Key StartItem StartBar EndItem EndBar Visible StartPos EndPos Color Style
Width ShowDir Text ... />
        <Link Key StartItem StartBar EndItem EndBar Visible StartPos EndPos Color Style
Width ShowDir Text ... />
        ...
      </Links>
    </Chart>
  - <Columns>
      <Column Caption Position Width HTMLCaption LevelKey DisplayFilterButton
DisplayFilterPatter FilterType ... />
      <Column Caption Position Width HTMLCaption LevelKey DisplayFilterButton
DisplayFilterPatter FilterType ... />
      ...
    </Columns>
  - <Items>
    - <Item Expanded ...>
        <Cell Value ValueFormat Images Image ... />
        <Cell Value ValueFormat Images Image ... />
        ...
      - <Bars>
          <Bar Name Start End Caption HAlignCaption VAlignCaption Key ... />
          <Bar Name Start End Caption HAlignCaption VAlignCaption Key ... />
          ...
```

```
        </Bars>
      - <Items>
        - <Item Expanded ...>
        - <Item Expanded ...>
        ....
        </Items>
      </Item>
    </Items>
  </Content>
```

method Grid.Scroll (Type as ScrollEnum, [ScrollTo as Variant])

Scrolls the control's content.

Type	Description
Type as ScrollEnum	A ScrollEnum expression that indicates type of scrolling being performed.
ScrollTo as Variant	A long expression that indicates the position where the control is scrolled when Type is exScrollVTo or exScrollHTo. If the ScrollTo parameter is missing, 0 value is used.

Use the Scroll method to scroll the control's content by code. Use the Scrollbars property specifies which scroll bars will be visible on the control. Use the ScrollPos property to get the control's scroll position. Use the EnsureVisibleItem method to ensure that a specified item fits the control's client area. 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. Use the ScrollPartVisible property to add buttons to the control's scrollbars. Use the Background property to change the visual appearance of the control's scrollbars.

If the Scroll(exScrollVTo) does not work please check if the ScrollBars property includes the exVScrollOnThumbRelease, and use a code like follows:

```
With Grid1
   .ScrollBars = .ScrollBars And Not exVScrollOnThumbRelease
   .Scroll exScrollVTo, 10000
   .ScrollBars = .ScrollBars Or exVScrollOnThumbRelease
End With
```

The code removes temporary the exVScrollOnThumbRelease flag from the ScrollBars property, performs the scrolling (jump to row 10000) , and restore back the exVScrollOnThumbRelease flag.

The following VB sample scrolls vertically the control line by line:

```
Private Sub Command1_Click()
   Grid1.Scroll exScrollDown
End Sub
```

The following VB sample scrolls the control's content to the top:

```
Private Sub Command1_Click()
   Grid1.Scroll exScrollVTo, 0
End Sub
```

The following VB sample scrolls the control's content to the first item (scrolls to the top):

```
Grid1.Scroll exScrollVTo, 0
```

The following C++ sample scrolls the control's content to the top:

```
m_grid.Scroll( 2 /*exScrollVTo*/, COleVariant( (long)0 ) );
```

The following C# sample scrolls the control's content to the top:

```
axGrid1.Scroll(EXGRIDLib.ScrollEnum.exScrollVTo, 0);
```

The following VB.NET sample scrolls the control's content to the top:

```
AxGrid1.Scroll(EXGRIDLib.ScrollEnum.exScrollVTo, 0)
```

The following VFP sample scrolls the control's content to the top:

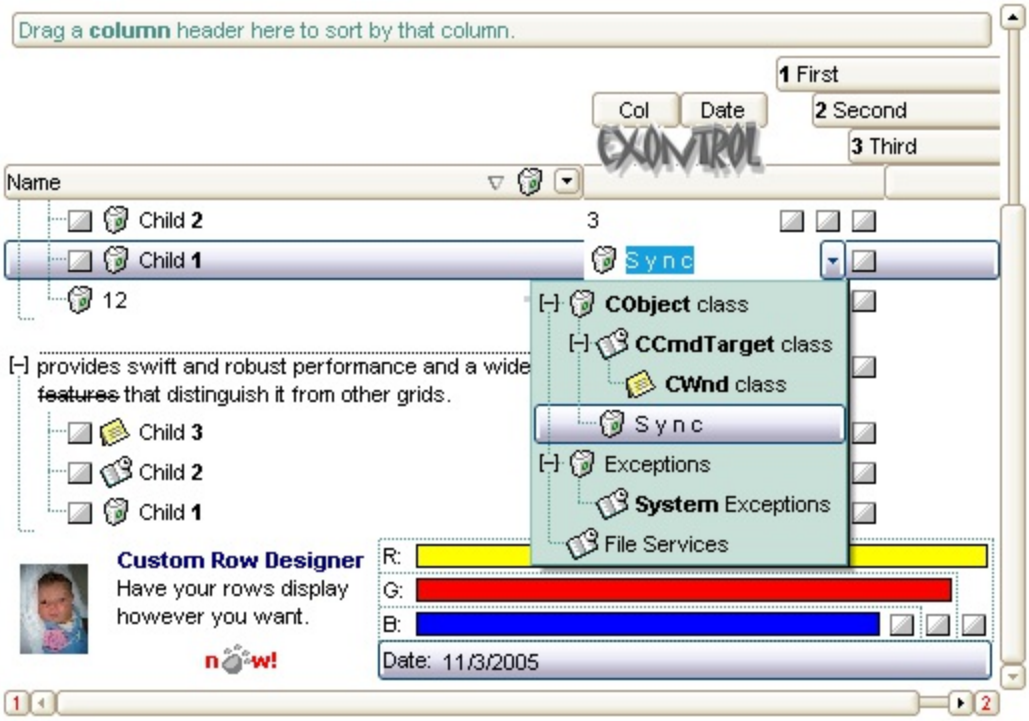
```
with thisform.Grid1
   .Scroll( 2, 0 ) && exScrollVTo
endwith
```

property Grid.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 indicates which scroll bars will be visible in the control.

By default, the control adds scroll bars when required. For instance, If the ColumnAutoResize property is False and the width of the visible columns exceeds the width of the control's client area, the control shows the horizontal scroll bar. Use the ScrollBars property to hide the control's scroll bars. If the ColumnAutoResize property is True, the control does not display the control's horizontal scroll bar. Use the ScrollBySingleLine property to let users scroll the control's content item by item. Use the ContinueColumnScroll property to specify whether the user scrolls the control's content column by column or pixel by pixel. 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. The ScrollBars property doesn't indicate whether the control displays a scroll bar. Instead, the WS_HSCROLL and WS_VSCROLL window styles indicate whether the window displays a scroll bar. Use the hWnd property to determine the handle of the control's window. Use the ScrollPartVisible property to add buttons to the control's scrollbars. Use the Background property to change the visual appearance of the control's scrollbars.



property Grid.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 Grid.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 Grid.ScrollBySingleLine as Boolean

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.

Type	Description
Boolean	A boolean expression that indicates whether the control scrolls the lines one by one.

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.

property Grid.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 Grid.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 Grid.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.

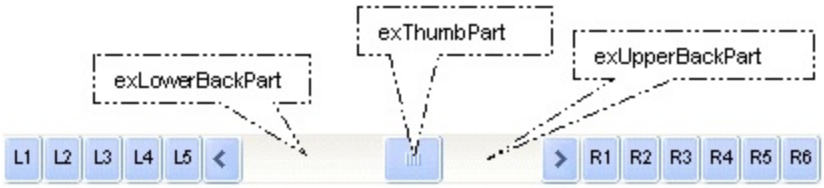
Any other literal between commas is ignored. If duplicate literals are found, the second is ignored, and so on. For instance, "t,l,r" indicates that the left/top and right/bottom buttons are displayed right/bottom after the thumb area.

property Grid.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 Grid1
   .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 AxGrid1
   .BeginUpdate()
   .ScrollBars = EXGRIDLib.ScrollBarsEnum.exDisableBoth
   .set_ScrollPartVisible(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exLeftB1Part Or EXGRIDLib.ScrollPartEnum.exRightB1Part, True)
   .set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exLeftB1Part, "<img></img>1")
   .set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.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 :

```
axGrid1.BeginUpdate();
axGrid1.ScrollBars = EXGRIDLib.ScrollBarsEnum.exDisableBoth;
axGrid1.set_ScrollPartVisible(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exLeftB1Part | EXGRIDLib.ScrollPartEnum.exRightB1Part, true);
axGrid1.set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exLeftB1Part , "<img></img>1");
axGrid1.set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exRightB1Part, "<img></img>2");
axGrid1.EndUpdate();
```

The following C++ sample adds up and down additional buttons to the control's vertical scroll bar :

```
m_grid.BeginUpdate();
m_grid.SetScrollBars( 15 /*exDisableBoth*/ );
m_grid.SetScrollPartVisible( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ | 32
/*exRightB1Part*/, TRUE );
m_grid.SetScrollPartCaption( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ , _T("<img>
</img>1") );
m_grid.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img></img>2")
);
m_grid.EndUpdate();
```

The following VFP sample adds up and down additional buttons to the control's vertical scroll bar :

```
With thisform.Grid1
   .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 Grid.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 Grid1
   .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 AxGrid1
```

```
.set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exHScroll,EXGRIDLib.ScrollPartEnum.exLov

.set_ScrollPartCaptionAlignment(EXGRIDLib.ScrollBarEnum.exHScroll,EXGRIDLib.ScrollPartE

.set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exHScroll,EXGRIDLib.ScrollPartEnum.exUp

.set_ScrollPartCaptionAlignment(EXGRIDLib.ScrollBarEnum.exHScroll,EXGRIDLib.ScrollPartE

   .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:

```
axGrid1.set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exHScroll,EXGRIDLib.ScrollPartEnu

axGrid1.set_ScrollPartCaptionAlignment(EXGRIDLib.ScrollBarEnum.exHScroll,EXGRIDLib.Scr

axGrid1.set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exHScroll,EXGRIDLib.ScrollPartEnu

axGrid1.set_ScrollPartCaptionAlignment(EXGRIDLib.ScrollBarEnum.exHScroll,EXGRIDLib.Scr

axGrid1.ColumnAutoResize = false;
axGrid1.Columns.Add(1.ToString());
axGrid1.Columns.Add(2.ToString());
axGrid1.Columns.Add(3.ToString());
axGrid1.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 'EXGRIDLib' for the library: 'ExGrid 1.0 Control Library'

   #import "ExGrid.dll"
   using namespace EXGRIDLib;
*/
EXGRIDLib::IGridPtr spGrid1 = GetDlgItem(IDC_GRID1)->GetControlUnknown();
spGrid1->PutScrollPartCaption(EXGRIDLib::exHScroll,EXGRIDLib::exLowerBackPart,L"left");
spGrid1-
>PutScrollPartCaptionAlignment(EXGRIDLib::exHScroll,EXGRIDLib::exLowerBackPart,EXGRID

spGrid1-
>PutScrollPartCaption(EXGRIDLib::exHScroll,EXGRIDLib::exUpperBackPart,L"right");
spGrid1-
>PutScrollPartCaptionAlignment(EXGRIDLib::exHScroll,EXGRIDLib::exUpperBackPart,EXGRID

spGrid1->PutColumnAutoResize(VARIANT_FALSE);
spGrid1->GetColumns()->Add(L"1");
spGrid1->GetColumns()->Add(L"2");
spGrid1->GetColumns()->Add(L"3");
spGrid1->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.Grid1
   .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)
```

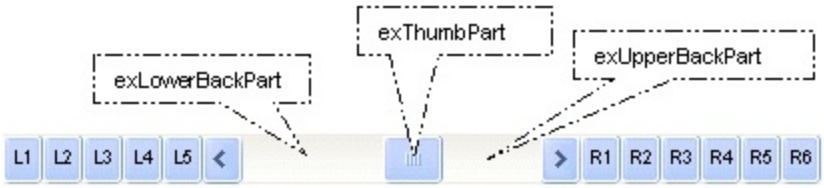
```
    .Columns.Add(4)
endwith
```

property Grid.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.

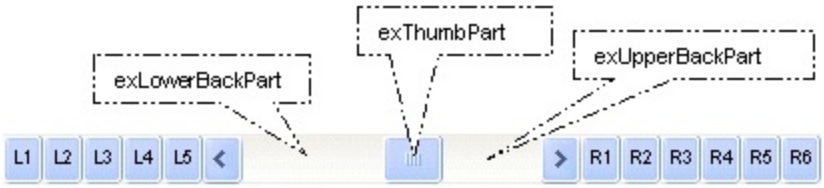


property Grid.ScrollPartVisible(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as Boolean

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 Grid1
   .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 AxGrid1
   .BeginUpdate()
   .ScrollBars = EXGRIDLib.ScrollBarsEnum.exDisableBoth
   .set_ScrollPartVisible(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exLeftB1Part Or EXGRIDLib.ScrollPartEnum.exRightB1Part, True)
   .set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exLeftB1Part, "<img></img>1")
   .set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.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 :

```
axGrid1.BeginUpdate();
axGrid1.ScrollBars = EXGRIDLib.ScrollBarsEnum.exDisableBoth;
axGrid1.set_ScrollPartVisible(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exLeftB1Part | EXGRIDLib.ScrollPartEnum.exRightB1Part, true);
axGrid1.set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exLeftB1Part , "<img></img>1");
axGrid1.set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exRightB1Part, "<img></img>2");
axGrid1.EndUpdate();
```

The following C++ sample adds up and down additional buttons to the control's vertical scroll bar :

```
m_grid.BeginUpdate();
m_grid.SetScrollBars( 15 /*exDisableBoth*/ );
m_grid.SetScrollPartVisible( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ | 32
/*exRightB1Part*/, TRUE );
m_grid.SetScrollPartCaption( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ , _T("<img>
</img>1") );
m_grid.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img></img>2")
);
m_grid.EndUpdate();
```

The following VFP sample adds up and down additional buttons to the control's vertical scroll bar :

```
With thisform.Grid1
   .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 Grid.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.

The following VB sample scrolls to the row 10,000:

```
With Grid1
   .ScrollPos(True) = 10000
End With
```

The following VB sample gets the cell's coordinates to let user aligns nicely a context popup menu:

```
Private Sub getCellPos(ByVal g As EXGRIDLibCtl.Grid, ByVal hItem As EXGRIDLibCtl.hItem,
ByVal nColumn As Long, X As Long, Y As Long)
   X = -g.ScrollPos(False)
   With g
      Dim c As EXGRIDLibCtl.Column
      For Each c In .Columns
         If (c.Visible) Then
            If (c.Position < .Columns(nColumn).Position) Then
               X = X + c.Width
            End If
         End If
      Next
      Y = 0
      If (.HeaderVisible) Then
         Y = Y + .HeaderHeight
```

```
    End If
    With .Items
      Dim i As EXGRIDLibCtl.hItem
      i = .FirstVisibleItem()
      While Not (i = hItem) And Not (i = 0)
        Y = Y + .ItemHeight(i)
        i = .NextVisibleItem(i)
      Wend
    End With
  End With
End Sub
```

The getCellPos method gets the x, y client coordinates of the cell (hItem, nColumn). The hItem indicates the handle of the item, and the nColumn indicates the index of the column. Use the ClientToScreen API function to convert the client coordinates to screen coordinates like bellow:

```
Private Type POINTAPI
    x As Long
    y As Long
End Type
Private Declare Function ClientToScreen Lib "user32" (ByVal hwnd As Long, lpPoint As POINTAPI) As Long
```

In VFP, use 1 instead .t. for changing the ScrollPos property like follows:

```
list1.object.scrollpos(1) = 123
```

instead

```
list1.scrollpos(.t.) = 123
```

In the following MouseDown handler the ItemFromPoint method determines the cell from the cursor. The sample displays an exPopupMenu control at the beginning of the cell, when user right clicks the cell:

```
Private Sub Grid1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
  If Button = 2 Then
    With Grid1
```

```
        Dim h As EXGRIDLibCtl.hItem, c As Long, hit As EXGRIDLibCtl.HitTestInfoEnum
        h = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
        If Not (h = 0) Then
            ' Selects the item when user does a right click
            Grid1.Items.SelectItem(h) = True
            ' Gets the client coordinates of the cell
            Dim xCell As Long, yCell As Long
            getCellPos Grid1, h, c, xCell, yCell
            ' Converts the client coordinates to the screen coordinates
            Dim p As POINTAPI
            p.X = xCell
            p.Y = yCell
            ClientToScreen Grid1.hwnd, p
            ' Displays the exPopupMenu control at specified position
            PopupMenu1.HAlign = EXPOPUPMENULibCtl.exLeft
            Debug.Print "You have selected " & PopupMenu1.Show(p.X, p.Y)
        End If
    End With
  End If
End Sub
```

property Grid.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 Grid.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 Grid1_OffsetChanged(ByVal Horizontal As Boolean, ByVal NewVal As Long)
   If (Not Horizontal) Then
      Grid1.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 AxGrid1_OffsetChanged(ByVal sender As System.Object, ByVal e As
AxEXGRIDLib._IGridEvents_OffsetChangedEvent) Handles AxGrid1.OffsetChanged
   If (Not e.horizontal) Then
      AxGrid1.set_ScrollToolTip(EXGRIDLib.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 OnOffsetChangedGrid1(BOOL Horizontal, long NewVal)
{
   if ( !Horizontal )
   {
      CString strFormat;
      strFormat.Format( _T("%i"), NewVal );
      m_grid.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 axGrid1_OffsetChanged(object sender,
AxEXGRIDLib._IGridEvents_OffsetChangedEvent e)
{
   if ( !e.horizontal )
      axGrid1.set_ScrollToolTip(EXGRIDLib.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.Grid1.ScrollToolTip(0) = "Record " + ltrim(str(newval))
EndIf
```

property Grid.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 Grid.SearchColumnIndex as Long

Retrieves or sets a value indicating the index of the column that is used by the auto search feature.

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

Use the SearchColumnIndex property to change the searching column. The control changes the searching column when the user clicks on a column or when the user presses the TAB key (in this case the UseTabKey property should be True). If the user starts typing characters in the searching column, the control selects the item that matches the typed characters. If you want to disable the auto search feature, you have to set the SearchColumnIndex property to -1. Use the MarkSearchColumn property to hide the marker of the searching column. If the searching column is moved, the focused column is moved too. Use the FocusColumnIndex property to get the focused column.

property Grid.SelBackColor as Color

Retrieves or sets a value that indicates the selection background color.

Type	Description
Color	A color expression that defines the selected items background color. The last 7 bits in the high significant byte of the color 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 SelForeColor and SelBackColor properties to define colors for the 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. The SelBackColor property may display transparent areas using EBN files. The SelForeColor property is applied only if it is different that the control's foreground color. Use the SingleSel property to specify whether the control supports single or multiple selection. Use the SelectItem property to programmatically select an item giving its handle. The SelectedItem and SelectCount properties get the collection of selected items. 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?

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 indicates the index of the skin that we want to use. The sample applies the "" to the selected item(s):

```
With Grid1
   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_grid.GetVisualAppearance().Add( 0x23,
COleVariant(_T("D:\\Temp\\ExGrid_Help\\selected.ebn")) );
m_grid.SetSelBackColor( m_grid.GetSelBackColor() | 0x23000000 );
m_grid.SetSelForeColor( 0 );
```

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

```
With AxGrid1
   With .VisualAppearance
      .Add(&H23, "D:\Temp\ExGrid_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):

```
axGrid1.VisualAppearance.Add(0x23, "D:\\Temp\\ExGrid_Help\\selected.ebn");
axGrid1.Template = "SelBackColor = 587202560";
```

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

```
With thisform.Grid1
   With .VisualAppearance
      .Add(35, "D:\Temp\ExGrid_Help\selected.ebn")
   EndWith
   .SelForeColor = RGB(0, 0, 0)
   .SelBackColor = .SelBackColor + 587202560
EndWith
```

How do I assign a new look for the selected item?

The component supports skinning parts of the control, including the selected item. Shortly, the idea is that identifier of the skin being added to the Appearance collection is stored in the first significant byte of property of the color type. In our case, we know that the SelBackColor property changes the background color for the selected item. This is what we need to change. In other words, we need to change the visual appearance for the selected item, and that means changing the background color of the selected item. So, the following code (blue code) changes the appearance for the selected item:

```
With Grid1
   .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 Grid1
   .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 Grid.SelBackMode as BackModeEnum

Retrieves or sets a value that indicates whether the selection is transparent or opaque.

Type	Description
BackModeEnum	A BackModeEnum expression that indicates how the selected items are painted.

By default, the SelBackMode property is exOpaque. Use the SelBackMode property to specify how the selection appears. Use the SelBackMode property to specify how the control displays the selection when the control has a picture on its background. Use the SelBackColor property to specify the selection background color. Use the SelForeColor property to specify the selection foreground color.

property Grid.SelectByDrag as Boolean

Specifies whether the user selects multiple items by dragging.

Type	Description
Boolean	A boolean expression that specifies whether the user may select multiple items by drag and drop.

By default, SelectByDrag property is True. Use the SelectByDrag property to disable selecting multiple items by dragging. The SelectByDrag property has effect only if the control supports multiple selection. The SingleSel property controls the number of items that the user may select. For instance, if the SingleSel property is True, the user can't select multiple items, and so a single item may be selected at the time. If the SingleSel property is False, the user can select multiple items using the mouse, keyboard or both. When the SelectByDrag property is True, the user may click the non text area to start select items by dragging. Use the SelectByDrag property on False when your control requires OLE drag and drop operations, like when you select multiple items and drag them to a new position. Use the OLEDropMode property to specify whether the OLE drag and drop operations inside the control is allowed. For instance, if the SelectByDrag and OLEDropMode properties are on, sometimes it is confused what control should do when user clicks and start to select items.

property Grid.SelectColumnIndex as Long

Retrieves or sets a value that indicates the index of the selected column, if the FullRowSelect property is False.

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

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 Grid.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 (creates an inner cell). The newly created cell is called inner cell. Use the FocusColumnIndex property to retrieve the index of the column that has the focus. 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 FocusColumnIndex and SelectColumnInner property to change the cell that has the focus inside.

The control fires the FocusChanged event when the user changes:

- the focused item
- the focused column or an inner cell gets the focus.

The following VB sample determines the focused cell:

```
Private Sub Grid1_FocusChanged()
  With Grid1
    Dim hFocusCell As EXGRIDLibCtl.HCELL
    hFocusCell = .Items.ItemCell(.Items.FocusItem, .FocusColumnIndex)
    Debug.Print "Focus = " & .Items.CellCaption(, hFocusCell) & " (" & hFocusCell & ")"
  End With
End Sub
```

The following VB sample determines the focused cell, if the control contains inner cells:

```
Private Sub Grid1_FocusChanged()
  With Grid1
    Dim hFocusCell As EXGRIDLibCtl.HCELL
    hFocusCell = .Items.ItemCell(.Items.FocusItem, .FocusColumnIndex)
    If (.SelectColumnInner > 0) Then
      ' Do we selected an inner cell?
      hFocusCell = .Items.InnerCell(, hFocusCell, .SelectColumnInner)
```

```
        End If
        Debug.Print "Focus = " & .Items.CellCaption(, hFocusCell) & " (" & hFocusCell & ")"
    End With
End Sub
```

The following C++ sample displays the focused cell:

```
#include "Items.h"
void OnFocusChangedGrid1()
{
    if ( IsWindow( m_grid.m_hWnd ) )
    {
        CItems items = m_grid.GetItems();
        COleVariant vtItem( items.GetFocusItem() ), vtColumn(
m_grid.GetFocusColumnIndex() );
        CString strFormat;
        strFormat.Format( "Focus on '%s'", V2S( &items.GetCellValue( vtItem, vtColumn ) ) );
        OutputDebugString( strFormat );
    }
}
```

The following C++ sample displays the focused cell, if the control contains inner cells:

```
#include "Items.h"
void OnFocusChangedGrid1()
{
    if ( IsWindow( m_grid.m_hWnd ) )
    {
        CItems items = m_grid.GetItems();
        COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
        COleVariant vtFocusCell( items.GetItemCell( items.GetFocusItem(),
COleVariant(m_grid.GetFocusColumnIndex())));
        if ( m_grid.GetSelectColumnInner() > 0 )
            vtFocusCell = items.GetInnerCell( vtMissing, vtFocusCell,
COleVariant(m_grid.GetSelectColumnInner()));
        CString strFormat;
        strFormat.Format( "Focus on '%s'", V2S( &items.GetCellValue( vtMissing, vtFocusCell )
) );
```

```
        OutputDebugString( strFormat );
    }
}
```

The following VB.NET sample displays the focused cell:

```
Private Sub AxGrid1_FocusChanged(ByVal sender As Object, ByVal e As System.EventArgs)
Handles AxGrid1.FocusChanged
    With AxGrid1.Items
        Debug.Print("Focus on '" & .CellValue(.FocusItem,
AxGrid1.FocusColumnIndex()).ToString() & "'")
    End With
End Sub
```

The following VB.NET sample displays the focused cell, if the control contains inner cells:

```
Private Sub AxGrid1_FocusChanged(ByVal sender As Object, ByVal e As System.EventArgs)
Handles AxGrid1.FocusChanged
    With AxGrid1.Items
        Dim focusCell As Object = .ItemCell(.FocusItem, AxGrid1.FocusColumnIndex)
        If (AxGrid1.SelectColumnInner > 0) Then
            focusCell = .InnerCell(Nothing, focusCell, AxGrid1.SelectColumnInner)
        End If
        Debug.Print("Focus on '" & .CellValue(Nothing, focusCell).ToString() & "'")
    End With
End Sub
```

The following C# sample displays the focused cell:

```
private void axGrid1_FocusChanged(object sender, EventArgs e)
{
    object focusValue = axGrid1.Items.get_CellValue(axGrid1.Items.FocusItem,
axGrid1.FocusColumnIndex);
    System.Diagnostics.Debug.WriteLine("Focus on '" + (focusValue != null ?
focusValue.ToString() : "" ) + "'");
}
```

The following C# sample displays the focused cell, if the control contains inner cells:

```
private void axGrid1_FocusChanged(object sender, EventArgs e)
{
   object focusCell = axGrid1.Items.get_ItemCell(axGrid1.Items.FocusItem,
axGrid1.FocusColumnIndex);
   if ( axGrid1.SelectColumnInner > 0 )
      focusCell = axGrid1.Items.get_InnerCell( null, focusCell, axGrid1.SelectColumnInner );
   object focusValue = axGrid1.Items.get_CellValue(null, focusCell);
   System.Diagnostics.Debug.WriteLine("Focus on '" + (focusValue != null ?
focusValue.ToString() : "" ) + "'");
}
```

The following VFP sample displays the focused cell:

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

with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   wait window nowait .CellCaption( 0, thisform.Grid1.FocusColumnIndex() )
endwith
```

property Grid.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. The SelectOnRelease property has no effect if the SingleSel property is False, and SelectByDrag property is True.

property Grid.SelForeColor as Color

Retrieves or sets a value that indicates the selection foreground color.

Type	Description
Color	A color expression that defines the selection foreground color.

Use the SelForeColor and SelBackColor properties to define colors for the 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 SelectItem property to programmatically select an item giving its handle. The SelectedItem and SelectCount properties get the collection of selected items. 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. The SelForeColor property is applied only if it is different that the control's foreground color.

property Grid.ShowFocusRect as Boolean

Retrieves or sets a value indicating whether the control draws a thin rectangle around the focused item.

Type	Description
Boolean	A boolean expression that indicates whether the marker for the focused cell is visible or hidden.

Use the ShowFocusRect property to hide the rectangle drawn around the focused item. Use the FocusItem property to get the focused item. Use the FocusColumnIndex property to determine the focused column. 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 Grid.ShowImageList as Boolean

Specifies whether the control's image list window is visible or hidden.

Type	Description
Boolean	A boolean expression that indicates whether the control's images list window is visible or hidden.

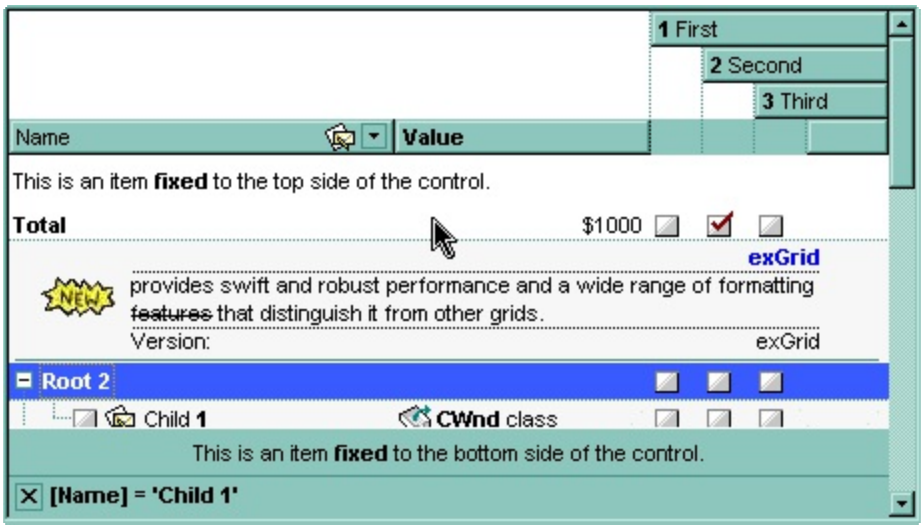
By default, the ShowImageList property is True. Use the ShowImageList property to hide the control's images list window. The control's images list window is visible only at design time. Use the Images method to associate an images list control to the tree control. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. Use the ReplaceIcon 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 Grid.ShowLockedItems as Boolean

Retrieves or sets a value that indicates whether the locked/fixed 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 CellValue property assign a value to a cell.



method Grid.ShowToolTip (ToolTip as String, [Title as Variant], [Alignment as Variant], [X as Variant], [Y as Variant])

Shows the specified tooltip at given position.

Type	Description
ToolTip as String	The ToolTip parameter can be any of the following: • NULL(BSTR) or "<null>"(string) to indicate that the tooltip for the object being hovered is not changed • A String expression that indicates the description of the tooltip, that supports built-in HTML format (adds, replaces or changes the object's tooltip)
Title as Variant	The Title parameter can be any of the following: • missing (VT_EMPTY, VT_ERROR type) or "<null>" (string) the title for the object being hovered is not changed. • A String expression that indicates the title of the tooltip (no built-in HTML format) (adds, replaces or changes the object's title)
Alignment as Variant	A long expression that indicates the alignment of the tooltip relative to the position of the cursor. If missing (VT_EMPTY, VT_ERROR) the alignment of the tooltip for the object being hovered is not changed. The Alignment parameter can be one of the following: • 0 - exTopLeft • 1 - exTopRight • 2 - exBottomLeft • 3 - exBottomRight • 0x10 - exCenter • 0x11 - exCenterLeft • 0x12 - exCenterRight • 0x13 - exCenterTop • 0x14 - exCenterBottom By default, the tooltip is aligned relative to the top-left corner (0 - exTopLeft).

X as Variant	Specifies the horizontal position to display the tooltip as one of the following: • missing (VT_EMPTY, VT_ERROR type), indicates that the tooltip is shown on its default position / current cursor position (ignored) • -1, indicates the current horizontal position of the cursor (current x-position) • a numeric expression that indicates the horizontal screen position to show the tooltip (fixed screen x-position) • a string expression that indicates the horizontal displacement relative to default position to show the tooltip (moved)
Y as Variant	Specifies the vertical position to display the tooltip as one of the following: • missing (VT_EMPTY, VT_ERROR type), indicates that the tooltip is shown on its default position / current cursor position (ignored) • -1, indicates the current vertical position of the cursor (current y-position) • a numeric expression that indicates the vertical screen position to show the tooltip (fixed screen y-position) • a string expression that indicates the vertical displacement relative to default position to show the tooltip (displacement)

Use the ShowToolTip method to display a custom tooltip at specified position or to update the object's tooltip, title or position. You can call the ShowToolTip method during the MouseMove event. Use the ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears. Use the ToolTipWidth property to specify the width of the tooltip window. Use the ToolTipFont property to change the tooltip's font. Use the Background(exToolTipAppearance) property indicates the visual appearance of the borders of the tooltips. Use the Background(exToolTipBackColor) property indicates the tooltip's background color. Use the Background(exToolTipForeColor) property indicates the tooltip's foreground color.

For instance:

- ShowToolTip(`<null>`,`<null>`,,`+8`,`+8`), shows the tooltip of the object moved relative

to its default position

- ShowToolTip(`<null>`,`new title`), adds, changes or replaces the title of the object's tooltip
- ShowToolTip(`new content`), adds, changes or replaces the object's tooltip
- ShowToolTip(`new content`,`new title`), shows the tooltip and title at current position
- ShowToolTip(`new content`,`new title`,,`+8`,`+8`), shows the tooltip and title moved relative to the current position
- ShowToolTip(`new content`,``,,128,128), displays the tooltip at a fixed position
- ShowToolTip(``,``), hides the tooltip

The ToolTip parameter supports the built-in HTML format like follows:

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

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

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

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

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

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

  gradient-center

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

  outlined

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

  shadow

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

  outline anti-aliasing

property Grid.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 support single or multiple selection.

The SingleSel property specifies whether the control support single or multiple selection. By default, the SingleSel property is True, and so only a single item can be selected. Use the SelectByDrag property to disable selecting multiple items by dragging. Use the FocusItem to retrieve the handle of the focused item. If the control supports single selection, the FocusItem property gets the handle of the selected item too. The SelectedItem and SelectCount properties get the collection of selected items. Use the SelectItem property to programmatically select an item giving its handle. The control fires SelectionChanged event when the selection is changed. Use the SelBackColor and SelForeColor properties to specify the background and foreground 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. The control doesn't support selection when using the virtual unbound mode. Use the SelectAll method to select all visible items, when the control supports multiple selection. The SelectPos property selects/unselects items by position. The Selection property selects/unselects items by index.

property Grid.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.

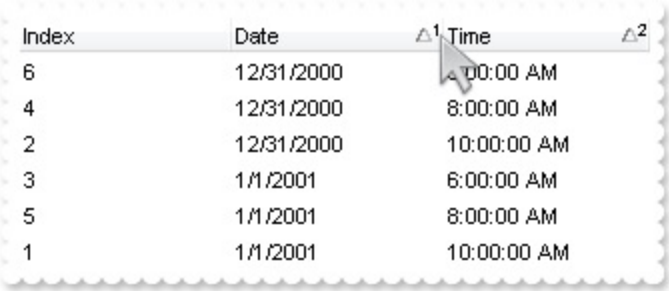
Use the SingleSort property to allow sorting by multiple columns. Sorting by a single column in the control is a simple matter of clicking on the column head. Sorting by multiple columns, however, is not so obvious. But it's actually quite easy. The user has two options to sort by multiple columns:

- 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.

If the control display no sort bar (SortBarVisible property is False), the column's header displays the position of the sorting order as shown bellow (multiple-columns sort):



property Grid.SortBarCaption as String

Specifies the caption being displayed on the control's sort bar when the sort bar contains no columns.

Type	Description
String	A String expression that indicates the caption of the control's sort bar.

The SortBarCaption property specifies the caption of the control's sort bar, when it contains no sorted columns. Use the SortBarVisible property to show the control's sort bar. Use the BackColorSortBar, BackColorSortBarCaption and ForeColorSortBar properties to specify colors for the control's sort bar. Use the SortBarHeight property to specify the height of the control's sort bar. Use the SortBarColumnWidth property to specify the width of the column in the control's sort bar. By default, the SortBarCaption property is "Drag a column header here to sort by that column.". Use the Font property to specify the control's font. Use the ItemBySortPosition property to access the columns in the control's sort bar.

The SortBarCaption property may include built-in HTML tags like follows:

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

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

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

Decode64Text/Encode64Text methods of the eXPrint can be used to decode/encode e64 fields.

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

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

loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.

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

gradient-center

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

outlined

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

shadow

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

property Grid.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.

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 Grid.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 Grid.SortBarVisible as Boolean

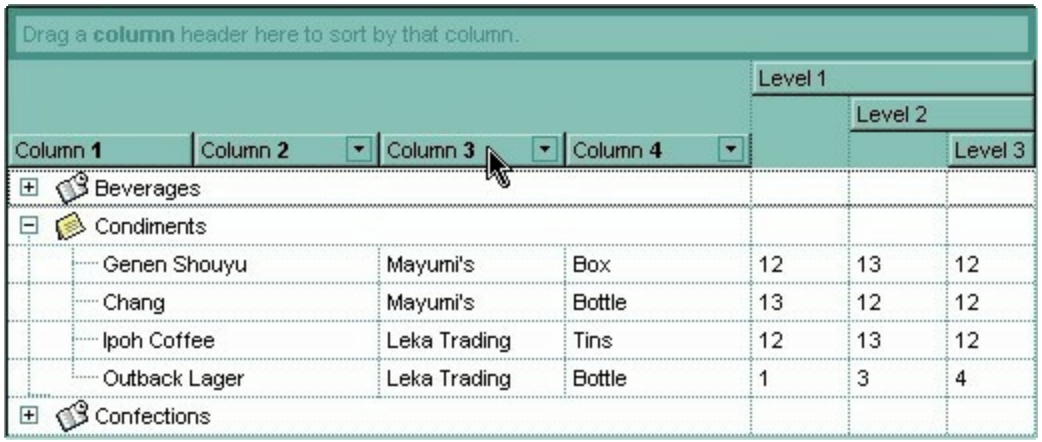
Retrieves or sets a value that indicates whether control's sort bar is visible or hidden.

Type	Description
Boolean	A boolean expression that indicates whether the sort bar is visible or hidden.

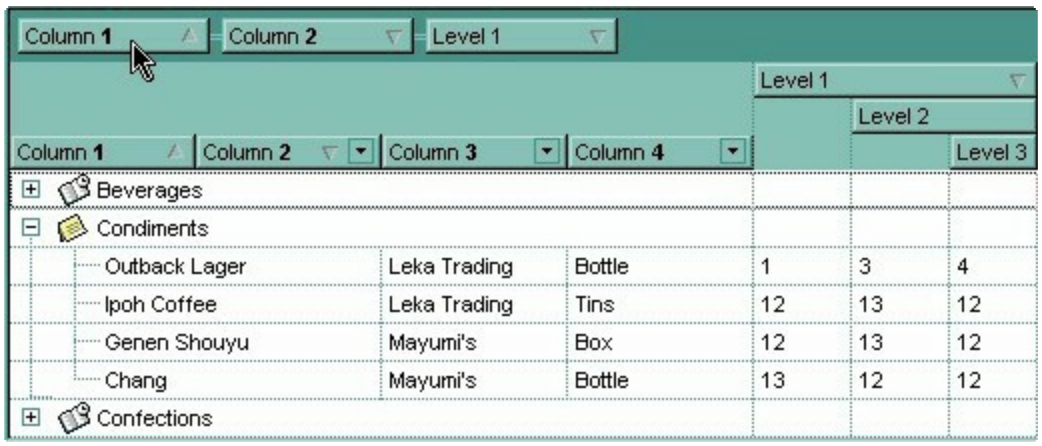
Use the SortBarVisible property to show the control's sort bar. By default, the SortBarVisible property is False. Use the SingleSort property to specify whether the control supports sorting by single or multiple columns. Sorting by a single column in the control is a simple matter of clicking on the 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 Grid.SortOnClick as SortOnClickEnum

Retrieves or sets a value that indicates whether the control automatically sorts the data when the user clicks on a column's caption.

Type	Description
SortOnClickEnum	A SortOnClickEnum expression that indicates the action that control takes whether the user clicks the column's header.

Use the SortOnClick property to 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. Use the SortChildren method to sort a column, at runtime. Use the DisplaySortIcon property to hide the sort icon if the column is sorted. Use the HeaderVisible property to show or hide the control's header. Use the BackColorHeader property to specify the header's background color. Use the AllowSizing property to disable resizing a column when user clicks the right margin of the column.

There are two methods to get the items sorted like follows:

- Using the SortOrder property of the Column object::

  ```
  Grid1.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).

  ```
  Grid1.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.

method Grid.StartBlockUndoRedo ()

Starts recording the UI operations as a block of undo/redo operations.

Type	Description

The StartBlockUndoRedo method starts recording the UI operations as a block on undo/redo operations. The method has effect only if the AllowUndoRedo property is True. The EndBlockUndoRedo method collects all undo/redo operations since StartBlockUndoRedo method was called and add them to the undo/redo queue as a block. This way the next call on a Undo operation, the entire block is restored, so all UI operations are restored. The EndBlockUndoRedo method must be called the same number of times as the StartBlockUndoRedo method was called. For instance, if you have called the StartBlockUndoRedo twice the EndBlockUndoRedo method must be called twice too, and the collected operations are added to the control's queue of undo/redo operations at the end.

property Grid.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 73 = 60,736 (60.40%)
Control: 1 x 20,440 = 20,440 (20.33%)
Column: 13 x 1,072 = 13,936 (13.86%)
Item: 64 x 72 = 4,608 (4.58%)
Items: 1 x 640 = 640 (0.64%)
Columns: 1 x 172 = 172 (0.17%)
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 73 = 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 129 = 107,328 (61.90%)
Control: 1 x 33,392 = 33,392 (19.26%)
Column: 13 x 1,760 = 22,880 (13.20%)
Item: 64 x 128 = 8,192 (4.72%)
Items: 1 x 1,224 = 1,224 (0.71%)
Columns: 1 x 320 = 320 (0.18%)
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 129 = 0 (0.00%)
CSmartVariant: 0 x 9 = 0 (0.00%)
```

property Grid.Template as String

Specifies the control's template.

Type	Description
String	A string expression that defines the control's template

The control's template uses the X-Script language to initialize the control's content. Use the Template property page of the control to update the control's Template property. Use the Template property to execute code by passing instructions as a string (template string). Use the ToTemplate property to generate the control's content to template format. Use the ExecuteTemplate property to get the result of executing a template script.

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

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

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

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

An instruction can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable **=** property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name*

of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))

- property**(** list of arguments **)** = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method**(** list of arguments **)** *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- object**.** property(list of arguments)**.**property(list of arguments)**.**... *The .(dot) character splits the object from its property. For instance, the Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The Template supports the following general functions:

- **RGB(**R,G,B**)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **CreateObject(**progID**)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier. For instance, the following code creates an ADOR.Recordset and pass it to the control using the DataSource property:*

The following sample loads the Orders table:

```
Dim rs
ColumnAutoResize = False
rs = CreateObject("ADOR.Recordset")
{
Open("Orders","Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program
Files\Exontrol\ExGrid\Sample\SAMPLE.MDB", 3, 3 )
}
DataSource = rs
```

property Grid.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 Grid.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 Grid.TooltipCellsColor as Color

Retrieves or sets a value that indicates the color used to mark the cells that have tool tips.

Type	Description
Color	A color expression that specifies the color used to mark the cells that have a tool tip associated.

The property has effect only if the MarkTooltipCells property is True. Use the CellToolTip property to assign a tooltip to a cell. Use the ToolTipWidth property to specify the width of the tooltip window. The control fires the ToolTip event when the column's tooltip is about to be displayed.

property Grid.ToolTipDelay as Long

Specifies the time in ms that passes before the ToolTip appears.

Type	Description
Long	A long expression that specifies the time in ms that passes before the ToolTip appears.

If the ToolTipDelay or ToolTipPopDelay property is 0, the control displays no tooltips. Use the ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the 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. Use the ToolTipFont property or <font> HTML element to assign a new font for tooltips.

property Grid.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. You can use the *<font>* HTML element, in the tooltip's description to assign a different font for portions of text.

property Grid.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 Grid.ToolTipWidth as Long

Specifies a value that indicates the width of the tooltip window, in pixels.

Type	Description
Long	A long expression that indicates the width of the tooltip window, in pixels.

Use the ToolTipWidth property to 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 Grid.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
AutoEdit = -1
```

In this case the ToTemplate property generates code only for the properties Appearance and AutoEdit, if they were changed to a different value.

If the DefaultTemplate parameters looks like:

```
Appearance
AutoEdit = -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)]
[3 = HTMLPicture("%KEY")=LoadPicture("%PICTURE")]
[1 = Images(%VALUE)]
Appearance = 2
ASCIILower = "abcdefghijklmnopqrstuvwxyzüéâäŕĺçęěčďîěôöňűůáíóúń"
ASCIIUpper = "ABCDEFGHIJKLMNOPQRSTUVWXYZÜÉÂÄŔĹÇĘĚČĎÎĚÔÖŇŰŮÁÍÓÚŃ"
AutoEdit = -1
AutoSearch = -1
BackColor = 2147483653
BackColorAlternate = 0
BackColorHeader = 2147483663
BackColorLevelHeader = 2147483663
BackColorLock = 2147483653
BackColorSortBar = 2147483664
BackColorSortBarCaption = 2147483663
Background(7) = 0
Background(6) = 0
Background(3) = 0
Background(2) = 0
```

Background(32) = 0
Background(8) = 0
Background(12) = 0
Background(11) = 0
Background(14) = 0
Background(13) = 0
Background(10) = 0
Background(9) = 0
Background(34) = 0
Background(33) = 0
Background(37) = 0
Background(36) = 0
Background(35) = 0
Background(5) = 0
Background(4) = 0
Background(1) = 0
Background(0) = 0
Background(404) = 0
Background(406) = 0
Background(407) = 0
Background(405) = 0
Background(384) = 0
Background(386) = 0
Background(387) = 0
Background(385) = 0
Background(396) = 0
Background(398) = 0
Background(399) = 0
Background(397) = 0
Background(392) = 0
Background(394) = 0
Background(395) = 0
Background(393) = 0
Background(388) = 0
Background(390) = 0
Background(391) = 0
Background(389) = 0

Background(400) = 0
Background(402) = 0
Background(403) = 0
Background(401) = 0
Background(324) = 0
Background(326) = 0
Background(327) = 0
Background(325) = 0
Background(511) = 0
Background(20) = 0
Background(17) = 0
Background(21) = 0
Background(15) = 0
Background(16) = 0
Background(25) = 0
Background(24) = 0
Background(23) = 0
Background(22) = 0
Background(276) = 0
Background(278) = 0
Background(279) = 0
Background(277) = 0
Background(264) = 0
Background(266) = 0
Background(267) = 0
Background(265) = 0
Background(268) = 0
Background(270) = 0
Background(271) = 0
Background(269) = 0
Background(260) = 0
Background(262) = 0
Background(263) = 0
Background(261) = 0
Background(256) = 0
Background(258) = 0
Background(259) = 0

Background(257) = 0
Background(272) = 0
Background(274) = 0
Background(275) = 0
Background(273) = 0
CauseValidateValue = 0
CheckImage(1) = 0
CheckImage(2) = 0
CheckImage(0) = 0
ColumnAutoResize = -1
ColumnsAllowSizing = 0
ContinueColumnScroll = -1
CountLockedColumns = 0
DefaultEditorOption(37) = 0
DefaultEditorOption(39) = 0
DefaultEditorOption(103) = 24
DefaultEditorOption(104) = "**7**,**8**,**9**,/,C\r\n**4**,**5**,**6**,*,1/x\r\n**1**,**2**,**3**,-,sqrt\r\n**0**,**+/-**,**.**,+,**=**"
DefaultEditorOption(102) = 24
DefaultEditorOption(101) = "Cannot divide by zero."
DefaultEditorOption(100) = -1
DefaultEditorOption(106) = ""
DefaultEditorOption(105) = ""
DefaultEditorOption(15) = 0
DefaultEditorOption(16) = 1
DefaultEditorOption(17) = 2
DefaultEditorOption(4) = 0
DefaultEditorOption(5) = -1
DefaultEditorOption(6) = -1
DefaultEditorOption(14) = -1
DefaultEditorOption(32) = 0
DefaultEditorOption(34) = 0
DefaultEditorOption(30) = "January February March April May June July August September October November December"
DefaultEditorOption(35) = -1
DefaultEditorOption(33) = -1
DefaultEditorOption(29) = "Today"
DefaultEditorOption(31) = "S M T W T F S"

DefaultEditorOption(47) = 0
DefaultEditorOption(23) = -1
DefaultEditorOption(28) = -1
DefaultEditorOption(46) = 46
DefaultEditorOption(36) = 0
DefaultEditorOption(50) = -2147483633
DefaultEditorOption(51) = 0
DefaultEditorOption(18) = 0
DefaultEditorOption(19) = 42
DefaultEditorOption(10) = 0
DefaultEditorOption(49) = -1
DefaultEditorOption(48) = 0
DefaultEditorOption(25) = -1
DefaultEditorOption(38) = 0
DefaultEditorOption(24) = -1
DefaultEditorOption(45) = 0
DefaultEditorOption(20) = -1
DefaultEditorOption(3) = -1
DefaultEditorOption(9) = -1
DefaultEditorOption(8) = 116
DefaultEditorOption(7) = 128
DefaultEditorOption(1) = 0
DefaultEditorOption(2) = 0
DefaultEditorOption(27) = -1
DefaultEditorOption(26) = -1
DefaultEditorOption(12) = 0
DefaultEditorOption(11) = -2147483635
DefaultEditorOption(13) = -1
DefaultEditorOption(21) = -1
DefaultEditorOption(52) = -1
DefaultEditorOption(44) = 100
DefaultEditorOption(43) = 0
DefaultEditorOption(42) = 1
DefaultEditorOption(41) = 64
DefaultEditorOption(40) = 1
DefaultEditorOption(22) = -1
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)"
Description(24) = "not"

```
Description(23) = "or"
DetectAddNew = 0
DetectDelete = 0
DrawGridLines = 0
Enabled = -1
EnsureOnSort = -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
FilterCriteria = ""
FocusColumnIndex = 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
MarkTooltipCells = 0
MarkTooltipCellsImage = 0
OLEDropMode = 0
PictureDisplay = 48
[255 = Picture = LoadPicture("%VALUE")]
PictureDisplayLevelHeader = 48
[256 = PictureLevelHeader = LoadPicture("%VALUE")]
RadioImage(0) = 0
RadioImage(-1) = 0
RClickSelect = 0
ReadOnly = 0
ScrollBars = 3
ScrollBySingleLine = 0
ScrollFont(0)
   Bold = 0
   Charset = 0
   Italic = 0
   Name = "Arial"
   Size = 6.75
   Strikethrough = 0
```

Underline = 0
Weight = 400
ScrollFont(1)
Bold = 0
Charset = 0
Italic = 0
Name = "Arial"
Size = 6.75
Strikethrough = 0
Underline = 0
Weight = 400
ScrollButtonWidth = -1
ScrollButtonHeight = -1
ScrollWidth = -1
ScrollHeight = -1
ScrollThumbSize(0) = -1
ScrollThumbSize(1) = -1
ScrollToolTip(0) = ""
ScrollToolTip(1) = ""
ScrollOrderParts(0) = ""
ScrollOrderParts(1) = ""
[4 = ScrollPartVisible(%BAR,%PART) = True]
[5 = ScrollPartEnable(%BAR,%PART) = True]
[6 = ScrollPartCaption(%BAR,%PART) = %VALUE]
SearchColumnIndex = 0
SelBackColor = 2147483661
SelBackMode = 0
SelectByDrag = -1
SelectColumnIndex = 0
SelectColumnInner = 0
SelForeColor = 2147483662
ShowFocusRect = -1
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
TooltipCellsColor = 16737585
ToolTipDelay = 500
ToolTipPopDelay = 5000
ToolTipWidth = 196
TreeColumnIndex = 0
UseTabKey = -1
ViewMode = 0
ViewModeOption(0,0) = 0
ViewModeOption(0,1) = 0
ViewModeOption(1,0) = 0
ViewModeOption(1,1) = 0
ViewModeOption(1,2) = 128
ViewModeOption(1,3) = 144
ViewModeOption(1,4) = "1/2/3/4/5/6/7"
ViewModeOption(1,5) = "0"
ViewModeOption(1,6) = -1
ViewModeOption(1,7) = 0
ViewModeOption(1,8) = 0
ViewModeOption(1,9) = 5
ViewModeOption(1,10) = 5
ViewModeOption(1,11) = -1
ViewModeOption(1,12) = 0
ConditionalFormats
   _NewEnum = Add("%Expression","%Key")
      ApplyTo = -1
      BackColor = 0
      Bold = 0
      Enabled = -1
      ForeColor = 0
      Italic = 0
      StrikeOut = 0
      Underline = 0
      Font
         Bold = 0
```

```
        Charset = 0
        Italic = 0
        Name = "Arial"
        Size = 8.25
        Strikethrough = 0
        Underline = 0
        Weight = 400
Columns
  _NewEnum = "%Caption"
    Alignment = 0
    AllowDragging = -1
    AllowSizing = -1
    AllowSort = -1
    AutoSearch = 0
    ComputedField = ""
    Data
    Def(4)
    Def(3) = 0
    Def(5)
    Def(32) = ""
    Def(2) = 0
    Def(0) = 0
    Def(1) = 0
    Def(16) = -1
    Def(17) = 0
    DefaultSortOrder = 0
    DisplayFilterButton = 0
    DisplayFilterDate = 0
    DisplayFilterPattern = -1
    DisplaySortIcon = -1
    Editor
      Appearance = 0
      ButtonWidth = 13
      DropDownAlignment = 0
      DropDownAutoWidth = -1
      DropDownMinWidth = 164
      DropDownRows = 7
```

DropDownVisible = -1
EditType = 0
Locked = 0
Mask = ""
MaskChar = 95
Numeric = 0
Option(37) = 0
Option(39) = 0
Option(103) = 24
Option(104) = "**7**,**8**,**9**,/,C\r\n**4**,**5**,**6**,*,1/x\r\n**1**,**2**,**3**,-,sqrt\r\n**0**,**+/-**,**.**,+,**=**"
Option(102) = 24
Option(101) = "Cannot divide by zero."
Option(100) = -1
Option(106) = ""
Option(105) = ""
Option(15) = 0
Option(16) = 1
Option(17) = 2
Option(4) = 0
Option(5) = -1
Option(6) = -1
Option(14) = -1
Option(32) = 0
Option(34) = 0
Option(30) = "January February March April May June July August September October November December"
Option(35) = -1
Option(33) = -1
Option(29) = "Today"
Option(31) = "S M T W T F S"
Option(47) = 0
Option(23) = -1
Option(28) = -1
Option(46) = 46
Option(36) = 0
Option(50) = -2147483633
Option(51) = 0

Option(18) = 0
Option(19) = 42
Option(10) = 0
Option(49) = -1
Option(48) = 0
Option(25) = -1
Option(38) = 0
Option(24) = -1
Option(45) = 0
Option(20) = -1
Option(3) = -1
Option(9) = -1
Option(8) = 116
Option(7) = 128
Option(1) = 0
Option(2) = 0
Option(27) = -1
Option(26) = -1
Option(12) = 0
Option(11) = -2147483635
Option(13) = -1
Option(21) = -1
Option(52) = -1
Option(44) = 100
Option(43) = 0
Option(42) = 1
Option(41) = 64
Option(40) = 1
Option(22) = -1
PartialCheck = 0
PopupAppearance = 6
[0 = AddButton(%BUTTON)]
[1 = AddItem(%ITEM)]
  [4 = ItemTooltip(%VALUE) = %TOOLTIP]
[2 = InsertItem(%ITEM)]
  [4 = ItemTooltip(%VALUE) = %TOOLTIP]
[3 = ExpandAll]

```
        [5 = UserEditor(%CONTROL,%LICENSE)]
      Enabled = -1
      FilterType = 0
      Filter = ""
      FilterList = 0
      FilterBarDropDownWidth = 1
      FireFormatColumn = 0
      FormatLevel = ""
      HeaderAlignment = 0
      HeaderBold = 0
      HeaderImage = 0
      HeaderImageAlignment = 0
      HeaderItalic = 0
      HeaderStrikeOut = 0
      HeaderUnderline = 0
      HeaderVertical = 0
      HTMLCaption = ""
      Key = ""
      LevelKey
      MaxWidthAutoResize = -1
      MinWidthAutoResize = 0
      PartialCheck = 0
      Position
      Selected = 0
      SortOrder = 0
      SortPosition = -1
      SortType = 0
      ToolTip = "..."
      Visible = -1
      Width = 64
      WidthAutoResize = 0
Items
   PathSeparator = "\"
   LockedItemCount(0) = 0
   LockedItemCount(2) = 0
   [1 = %H = LockedItem(%A,%I)]
      [10 = CellValue(%H,%C) = %VALUE]
```

```
[11 = CellImage(%H,%C) = %VALUE]
[12 = CellSingleLine(%H,%C) = %VALUE]
[13 = CellValueFormat(%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]
[22 = CellEditorVisible(%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]
[35 = CellData(%H,%C) = %VALUE]
[110 = CellEditor(%H,%C)]
   Appearance = 0
   ButtonWidth = 13
   DropDownAlignment = 0
   DropDownAutoWidth = -1
   DropDownMinWidth = 164
   DropDownRows = 7
   DropDownVisible = -1
   EditType = 0
   Locked = 0
   Mask = ""
   MaskChar = 95
```

Numeric = 0
Option(37) = 0
Option(39) = 0
Option(103) = 24
Option(104) = "**7**,**8**,**9**,/,C\r\n**4**,**5**,**6**,*,1/x\r\n**1**,**2**,**3**,-,sqrt\r\n**0**,**+/-**,**.**,+,**=**"
Option(102) = 24
Option(101) = "Cannot divide by zero."
Option(100) = -1
Option(106) = ""
Option(105) = ""
Option(15) = 0
Option(16) = 1
Option(17) = 2
Option(4) = 0
Option(5) = -1
Option(6) = -1
Option(14) = -1
Option(32) = 0
Option(34) = 0
Option(30) = "January February March April May June July August September October November December"
Option(35) = -1
Option(33) = -1
Option(29) = "Today"
Option(31) = "S M T W T F S"
Option(47) = 0
Option(23) = -1
Option(28) = -1
Option(46) = 46
Option(36) = 0
Option(50) = -2147483633
Option(51) = 0
Option(18) = 0
Option(19) = 42
Option(10) = 0
Option(49) = -1
Option(48) = 0

```
    Option(25) = -1
    Option(38) = 0
    Option(24) = -1
    Option(45) = 0
    Option(20) = -1
    Option(3) = -1
    Option(9) = -1
    Option(8) = 116
    Option(7) = 128
    Option(1) = 0
    Option(2) = 0
    Option(27) = -1
    Option(26) = -1
    Option(12) = 0
    Option(11) = -2147483635
    Option(13) = -1
    Option(21) = -1
    Option(52) = -1
    Option(44) = 100
    Option(43) = 0
    Option(42) = 1
    Option(41) = 64
    Option(40) = 1
    Option(22) = -1
    PartialCheck = 0
    PopupAppearance = 6
    [0 = AddButton(%BUTTON)]
    [1 = AddItem(%ITEM)]
      [4 = ItemTooltip(%VALUE) = %TOOLTIP]
    [2 = InsertItem(%ITEM)]
      [4 = ItemTooltip(%VALUE) = %TOOLTIP]
    [3 = ExpandAll]
    [5 = UserEditor(%CONTROL,%LICENSE)]
  [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]
[1014 = ItemData(%H) = %VALUE]
[0 = %H = %ADD(%VALUE)]
[10 = CellValue(%H,%C) = %VALUE]
[11 = CellImage(%H,%C) = %VALUE]
[12 = CellSingleLine(%H,%C) = %VALUE]
[13 = CellValueFormat(%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]
[22 = CellEditorVisible(%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]
[35 = CellData(%H,%C) = %VALUE]

[110 = CellEditor(%H,%C)]
Appearance = 0
ButtonWidth = 13
DropDownAlignment = 0
DropDownAutoWidth = -1
DropDownMinWidth = 164
DropDownRows = 7
DropDownVisible = -1
EditType = 0
Locked = 0
Mask = ""
MaskChar = 95
Numeric = 0
Option(37) = 0
Option(39) = 0
Option(103) = 24
Option(104) = "**7**,**8**,**9**,/,C\r\n**4**,**5**,**6**,*,1/x\r\n**1**,**2**,**3**,-,sqrt\r\n**0**,**+/-**,**.**,+,**=**"
Option(102) = 24
Option(101) = "Cannot divide by zero."
Option(100) = -1
Option(106) = ""
Option(105) = ""
Option(15) = 0
Option(16) = 1
Option(17) = 2
Option(4) = 0
Option(5) = -1
Option(6) = -1
Option(14) = -1
Option(32) = 0
Option(34) = 0
Option(30) = "January February March April May June July August September October November December"
Option(35) = -1
Option(33) = -1
Option(29) = "Today"
Option(31) = "S M T W T F S"

Option(47) = 0
Option(23) = -1
Option(28) = -1
Option(46) = 46
Option(36) = 0
Option(50) = -2147483633
Option(51) = 0
Option(18) = 0
Option(19) = 42
Option(10) = 0
Option(49) = -1
Option(48) = 0
Option(25) = -1
Option(38) = 0
Option(24) = -1
Option(45) = 0
Option(20) = -1
Option(3) = -1
Option(9) = -1
Option(8) = 116
Option(7) = 128
Option(1) = 0
Option(2) = 0
Option(27) = -1
Option(26) = -1
Option(12) = 0
Option(11) = -2147483635
Option(13) = -1
Option(21) = -1
Option(52) = -1
Option(44) = 100
Option(43) = 0
Option(42) = 1
Option(41) = 64
Option(40) = 1
Option(22) = -1
PartialCheck = 0

```
        PopupAppearance = 6
        [0 = AddButton(%BUTTON)]
        [1 = AddItem(%ITEM)]
          [4 = ItemTooltip(%VALUE) = %TOOLTIP]
        [2 = InsertItem(%ITEM)]
          [4 = ItemTooltip(%VALUE) = %TOOLTIP]
        [3 = ExpandAll]
        [5 = UserEditor(%CONTROL,%LICENSE)]
      [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]
      [1014 = ItemData(%H) = %VALUE]
      [1015 = ExpandCard(%H) = %VALUE]
[2 = ApplyFilter]
ScrollPos(0) = 0
ScrollPos(-1) = 0
[0 = EndUpdate]
```

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.

Let's say that you need only the items creating on the template like follows:

```
BeginUpdate
Items
{
```

```
    Dim h0
    h0 = AddItem("Number")
    CellValue(h0,1) = "Is Not"
    CellValue(h0,2) = "aaa"
    CellData(h0,0) = 2
    CellData(h0,1) = 18
    CellData(h0,2) = "aaa"
}
EndUpdate
```

In this case the DefaultTemplate parameter should be:

```
[0 = BeginUpdate]
Items
    [0 = %H = %ADD(%VALUE)]
        [10 = CellValue(%H,%C) = %VALUE]
        [35 = CellData(%H,%C) = %VALUE]
[0 = EndUpdate]
```

in VB this is translated to:

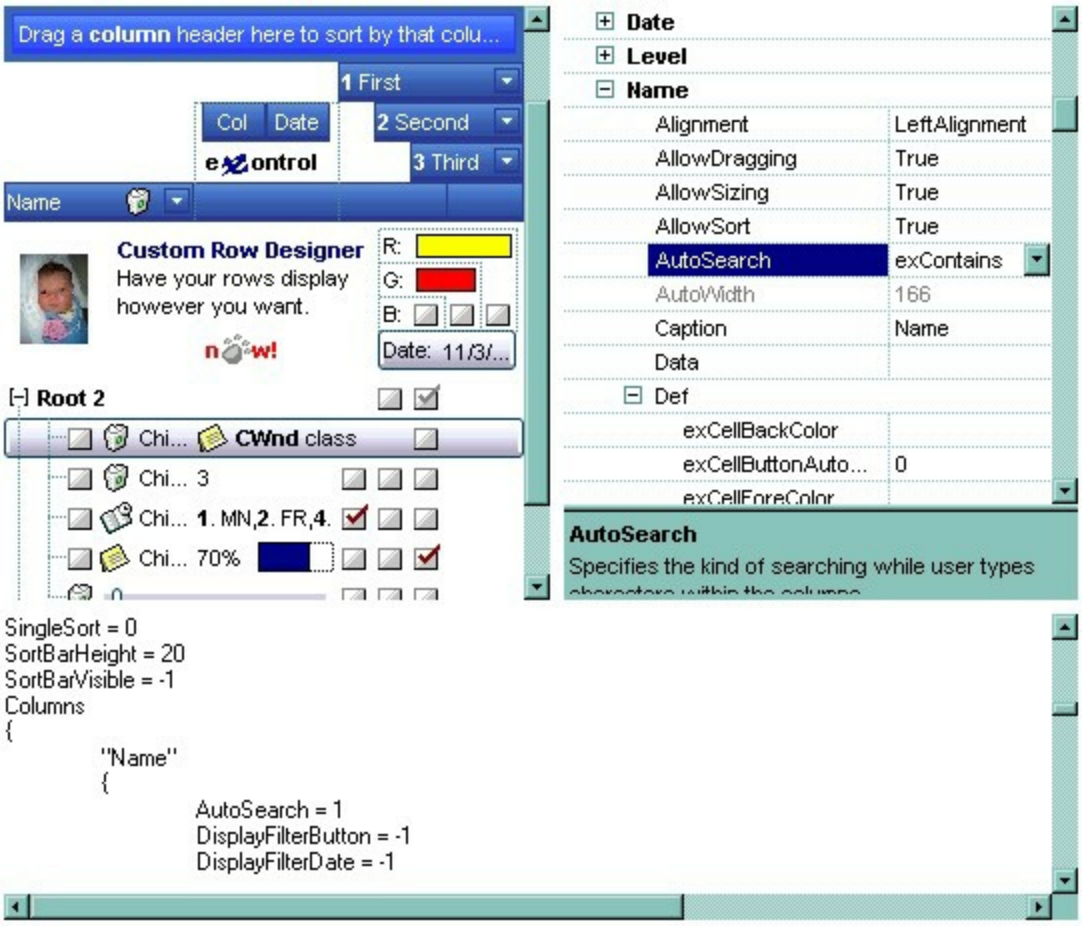
```
Dim dt As String
dt = dt + "[0 = BeginUpdate]" + vbCrLf
dt = dt + "Items" + vbCrLf
dt = dt + vbTab + "[0 = %H = %ADD(%VALUE)]" + vbCrLf
dt = dt + vbTab + vbTab + "[10 = CellValue(%H,%C) = %VALUE]" + vbCrLf
dt = dt + vbTab + vbTab + "[35 = CellData(%H,%C) = %VALUE]" + vbCrLf
dt = dt + "[0 = EndUpdate]"
Debug.Print Grid1.ToTemplate(dt)
```

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
```

In other words, the DefaultTemplate parameter holds the default values for properties in the control, including inside objects. Using the ToTemplate property and the ExPropertiesList browser you can write template files easily.



property Grid.TreeColumnIndex as Long

Retrieves or sets a value that indicates the index of column where the hierarchy lines are displayed.

Type	Description
Long	A long expression that indicates the index of column that displays the control's hierarchy.

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. Use the InsertItem property to insert child items.

property Grid.UnboundHandler as IUnboundHandler

Specifies the control's unbound handler.

Type	Description
IUnboundHandler	An object that implements the IUnboundHandler interface.

The control supports unbound mode. In unbound mode, the user is responsible for retrieving items. The unbound mode and virtual unbound modes were provided to let user displays large number of items. In order to let the control works in unbound mode, the user has to implement the IUnboundHandler notification interface. Use the VirtualMode property to run the control in virtual mode. Use the RemoveAllItems method to remove all items from the control, after setting the UnboundHandler property to nothing. Use the BeginUpdate and EndUpdate methods, or Refresh method after setting the UnboundHandler property, to reflect the changes in the control's client area.

If the VirtualMode property is True and the DataSource property is set (not empty), the control provides an internal object that implements the IUnboundHandler interface to provide data for the control from the data source.

The following VB sample shows how to implement the IUnboundHandler interface:

```
Option Explicit
Implements IUnboundHandler

Private Sub Form_Load()
  With Grid1
    .BeginUpdate
      .FullRowSelect = False

      ' Adds two columns
      With .Columns
        Dim i As Long
        For i = 0 To 1
          With .Add("Column " & i + 1).Editor
            .EditType = EditTypeEnum.EditType
          End With
        Next
      End With
       Set .UnboundHandler = Me
```

```
        .EndUpdate
    End With
End Sub

Private Property Get IUnboundHandler_ItemsCount(ByVal Source As Object) As Long
    ' The control requires the number of items. the Set .UnboundHandler = Me invokes the
IUnboundHandler_ItemsCount method
    IUnboundHandler_ItemsCount = 16
End Property

Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As Object,
ByVal ItemHandle As Long)
    ' The control requires an item
    With Grid1.Items
        .CellValue(ItemHandle, 0) = Index
        .CellValue(ItemHandle, 1) = Index + 1
    End With
End Sub
```

Running the UnboundMode in VFP 7.0 or greater

1. Create a PRG file, say, CLASS1.PRG to define a class to implement the IUnboundHandler interface:

```
define class UnboundHandler as custom
    implements IUnboundHandler in "ExGrid.dll"

    function IUnboundHandler_get_ItemsCount(Source)
        return 10000000   && we are going to have that many virtual items
    endfunc

    function IUnboundHandler_ReadItem(Index, Source, ItemHandle)
        With Source.Items
        .DefaultItem = ItemHandle
                .CellValue(0, 0) = 'Virtual Item ' + transform(Index+1)    && in this example,
just set text in Column 0
        EndWith
    endfunc
```

```
enddefine
```

2. Set up ExGrid's properties as follows (in the example, it is done by Form1.Init)

```
with thisform.Grid1
   .Columns.Add("Column 0")
   .UnboundHandler = newobject('UnboundHandler', 'class1.prg')
endwith
```

3. You can also use Virtual Mode to scan a table, say, MyCursor, in natural order.

```
function IUnboundHandler_get_ItemsCount(Source)
   return reccount('MyCursor')
endfunc

function IUnboundHandler_ReadItem(Index, Source, ItemHandle)
   select MyCursor
   go Index+1
   With Source.Items
      .DefaultItem = ItemHandle
         .CellValue(0, 0) = 'Virtual Item ' + MyCursor.Field1
   EndWith
endfunc
enddefine
```

Please check also the VirtualMode property that includes multiple samples. The setup program installs a sample VC\UnboundMode, for C++ version.

method Grid.Undo ()

Performs the last Undo operation.

Type	Description

Call the Undo method to Undo the last control operation. The Undo method have effect only if the AllowUndoRedo property is True. The CTRL+Z performs the last undo operation, while the CTRL+Y redoes the next action in the control's Redo queue. The Redo redoes the next action in the control's redo queue. The CanUndo property retrieves a value that indicates whether the control may perform the last Undo operation. The CanRedo property retrieves a value that specifies whether the control can execute the next operation in the control's Redo queue. The URChange(exUndo/exRedo) event notifies your application whenever an Undo/Redo operation is performed.

The records of the Undo/Redo queue may contain actions in the following format:

- "**AddItem**;ITEMINDEX", indicates that a new item has been created
- "**RemoveItem**;ITEMINDEX", indicates that an item has been removed
- "**ChangeItemPos**;ITEMINDEX", indicates that an item changes its position or / and parent
- "**ChangeCellValue**;ITEMINDEX;CELLINDEX", indicates that the cell's value has been changed
- "**ChangeCellState**;ITEMINDEX;CELLINDEX", indicates that the cell's state has been changed

Also, the Undo/Redo queue may include:

- "**StartBlock**", specifies that a block of operations begins
- "**EndBlock**", specifies that a block of operations ends

The UndoListAction property lists the Undo actions that can be performed in the control. Use the UndoRemoveAction method to remove the last actions from the undo queue.

property Grid.UndoListAction ([Action as Variant], [Count as Variant]) as String

Lists the Undo actions that can be performed in the control.

Type	Description
Action as Variant	[optional] A long expression that specifies the action being listed. If missing or -1, all actions are listed. The Action parameter can be one of the following: • exListUndoRedoAddItem(13) ~ "**AddItem**;ITEMINDEX", indicates that a new item has been created • exListUndoRedoRemoveItem(14) ~ "**RemoveItem**;ITEMINDEX", indicates that an item has been removed • exListUndoRedoChangeItemPos(15) ~ "**ChangeItemPos**;ITEMINDEX", indicates that an item changes its position or / and parent • exListUndoRedoChangeCellValue(16) ~ "**ChangeCellValue**;ITEMINDEX;CELLINDEX", indicates that the cell's value has been changed • exListUndoRedoChangeCellState(17) ~ "**ChangeCellState**;ITEMINDEX;CELLINDEX", indicates that the cell's state has been changed For instance, UndoListAction(13) shows only AddItem actions in the undo stack.
Count as Variant	[optional] A long expression that indicates the number of actions being listed. If missing or -1, all actions are listed. For instance, UndoListAction(13,1) shows only the last AddItem action being added to the undo stack
String	A String expression that lists the Undo actions that may be performed.

Use the UndoListAction property to show the list of actions that the user may perform by doing Undo operations. The URChange(exUndo/exRedo) event notifies your application whenever an Undo/Redo operation is performed. For instance, the URChange(exUndoRedoUpdate) notifies whether a new operation is added/removed from the undo/redo queue. Use the UndoRemoveAction method to remove the last actions from the undo queue. The RedoListAction property lists the Redo actions that can be performed

in the control. The CanUndo property specifies whether an undo operation can be performed if CTRL+Z key is pressed.

The records of the Undo/Redo queue may contain actions in the following format:

- "**AddItem**;ITEMINDEX", indicates that a new item has been created
- "**RemoveItem**;ITEMINDEX", indicates that an item has been removed
- "**ChangeItemPos**;ITEMINDEX", indicates that an item changes its position or / and parent
- "**ChangeCellValue**;ITEMINDEX;CELLINDEX", indicates that the cell's value has been changed
- "**ChangeCellState**;ITEMINDEX;CELLINDEX", indicates that the cell's state has been changed

Also, the Undo/Redo queue may include:

- "**StartBlock**", specifies that a block of operations begins
- "**EndBlock**", specifies that a block of operations ends

Each action is on a single line, and each field is separated by ; character. The lines are separated by "\r\n" characters (vbCrLf in VB).

The following VB sample splits the UndoListAction value and adds each action to a listbox control:

```
List1.Clear
Dim s() As String
s = Split(Grid1.UndoListAction, vbCrLf)
For i = LBound(s) To UBound(s)
        List1.AddItem s(i)
Next
```

property Grid.UndoRedoQueueLength as Long

Gets or sets the maximum number of Undo/Redo actions that may be stored to the control's queue.

Type	Description
Long	A Long expression that specifies the length of the Undo/Redo queue. If -1, the queue is unlimited, 0 allows no entries in the Undo/Redo queue.

By default, the UndoRedoQueueLength property is -1. Use the UndoRedoQueueLength property to specify the number of entries that Undo/Redo queue may store. For instance, if the UndoRedoQueueLength property is 1, the control retains only the last control operation. Changing the UndoRedoQueueLength property may change the current Undo/Redo queue based on the new length. The length being specified, does not affect the blocks in the queue. A block may hold multiple Undo/Redo actions. Use the GroupUndoRedoActions method to group two or more entries in the Undo/Redo queue in a single block, so when a next Undo/Redo operation is performed, multiple actions may occur. For instance, moving several bars in the same time (multiple bars selection) is already recorded as a single block.

method Grid.UndoRemoveAction ([Action as Variant], [Count as Variant])

Removes the last the undo actions that can be performed in the control.

Type	Description
Action as Variant	[optional] A long expression that specifies the action being removed. If missing or -1, all actions are removed from the undo queue. The Action parameter can be one of the following: • exListUndoRedoAddItem(13) ~ "**AddItem**;ITEMINDEX", indicates that a new item has been created • exListUndoRedoRemoveItem(14) ~ "**RemoveItem**;ITEMINDEX", indicates that an item has been removed • exListUndoRedoChangeItemPos(15) ~ "**ChangeItemPos**;ITEMINDEX", indicates that an item changes its position or / and parent • exListUndoRedoChangeCellValue(16) ~ "**ChangeCellValue**;ITEMINDEX;CELLINDEX", indicates that the cell's value has been changed • exListUndoRedoChangeCellState(17) ~ "**ChangeCellState**;ITEMINDEX;CELLINDEX", indicates that the cell's state has been changed For instance, UndoRemoveAction(13) removes only AddItem actions in the undo stack.
Count as Variant	[optional] A long expression that indicates the number of actions to be removed. If missing or -1, all actions are removed. For instance, UndoRemoveAction(13,1) removes only the last AddItem action from the undo stack

Use the UndoRemoveAction method to remove the last action from the undo queue. Use the UndoRemoveAction() (with no parameters) to remove all undo actions. The UndoListAction property retrieves the list of actions that an undo operation can perform. The RedoRemoveAction method removes the first action to be performed if the Redo method is invoked.

The records of the Undo/Redo queue may contain actions in the following format:

- "**AddItem**;ITEMINDEX", indicates that a new item has been created
- "**RemoveItem**;ITEMINDEX", indicates that an item has been removed
- "**ChangeItemPos**;ITEMINDEX", indicates that an item changes its position or / and parent
- "**ChangeCellValue**;ITEMINDEX;CELLINDEX", indicates that the cell's value has been changed
- "**ChangeCellState**;ITEMINDEX;CELLINDEX", indicates that the cell's state has been changed

Also, the Undo/Redo queue may include:

- "**StartBlock**", specifies that a block of operations begins
- "**EndBlock**", specifies that a block of operations ends

method Grid.Ungroup ()

Ungroups the columns, if they have been previously grouped.

Type	Description

The Ungroup method removes the grouping items from the control's list. The AllowGroupBy property specifies whether the control supports Group-By feature. The Ungroup method has no effect if the AllowGroupBy property is False, or no columns is grouped. The Group method forces the control to re-group the items. During execution any of these methods, the IsGrouping property returns True. You can call the SortOrder property to sort and group by specified column. Use the SortType property to determine the way how the column is sorted. The AddGroupItem event is fired when a new grouping items is added to the control's list. *You can use the AddGroupItem event, to add headers or footers during grouping, customize the aggregate formula to be displayed on different columns, while dropping a column to the sortbar.* The Column.AllowGroupBy property may be used to prevent grouping a specific column. The AllowSort property indicates whether the user can sort a column by clicking the column's header.

property Grid.UseTabKey as Boolean

Retrieves or sets a value indicating whether the control uses tab key for changing the searching column.

Type	Description
Boolean	A boolean expression indicating whether the control uses tab key for changing the searching column.

By default, the UseTabKey property is True. The UseTabKey property specifies whether the control uses the TAB key to change the searching column. If the UseTabKey property is False, the TAB key is used to navigate through the form's controls. Use the SearchColumnIndex property to specify the index of the searching column.

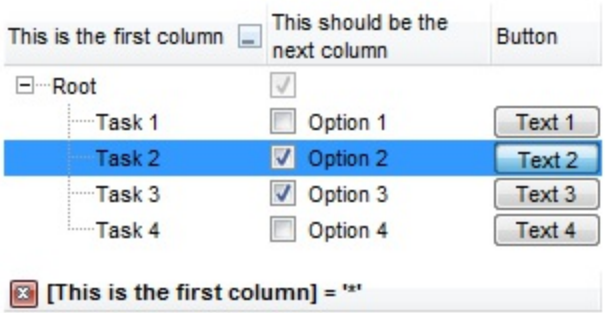
property Grid.UseVisualTheme as UIVisualThemeEnum

Specifies whether the control uses the current visual theme to display certain UI parts.

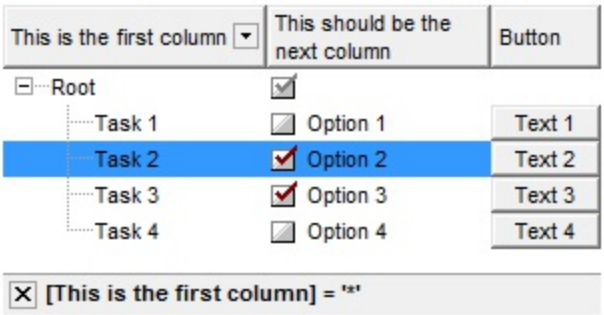
Type	Description
UIVisualThemeEnum	An UIVisualThemeEnum expression that specifies which UI parts of the control are shown using the current visual theme.

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 Grid.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 Grid.ViewMode as ViewModeEnum

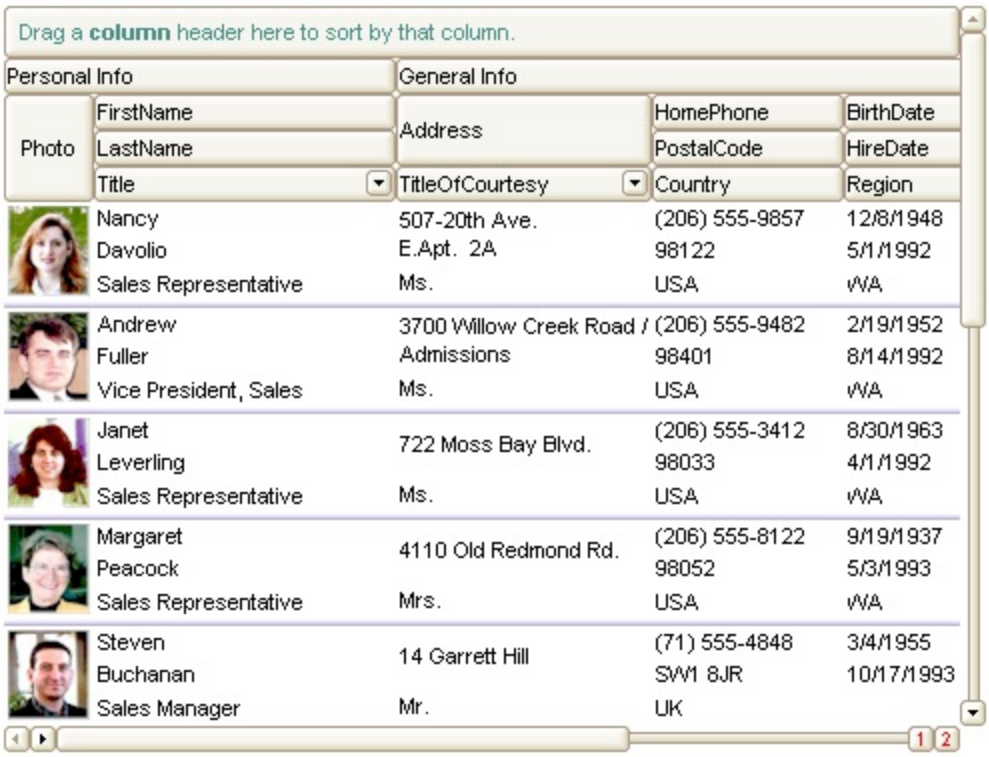
Specifies how the data is displayed on the control's view.

Type	Description
ViewModeEnum	A ViewModeEnum expression that indicates the way how data is displayed on the control's view.

By default, the ViewMode property is TableView. Currently, the control supports TableView mode and CardView mode. Use the Add method to add new columns to the control. Use the AddItem method to add new items/cards to your control. Use the PutItems, DataSource method and related to load or bind the control to a data set.

The TableView mode shows the items as rows in a table. The user can customize the layout of the fields in the row using the CellFormatLevel property.

The image below shows an example of TableView mode:



The CardView mode represents the view displaying data using cards. The user can customize the layout of the fields in the card or in the title of the card using the ViewModeOption(exCardViewFormat) ViewModeOption(exCardViewTitleFormat) properties. The control allows displaying cards from left to right or top to bottom, auto-arranging, resizing cards, expand or collapse cards support and much more.

The image below shows an example of CardView mode:

Nancy Davolio

Andrew Fuller

Janet Leverling

Margaret Peacock

Steven Buchanan

Buchanan
Steven
☑
Sales Manager
Mr.
3/4/1955
10/17/1993
14 Garrett Hill
London

SW1 8JR
UK
(71) 555-4848
3453

Michael Suyama

Suyama
Michael

Laura Callahan

Callahan
Laura
☐
Inside Sales Coordinator
Ms.
1/9/1958
3/5/1994
4726 - 11th Ave. N.E.
Seattle
WA
98105
USA
(206) 555-1189
2344

Anne Dodsworth

Dodsworth
Anne
☐
Sales Representative
Ms.
1/27/1966
11/15/1994

property Grid.ViewModeOption(Mode as ViewModeEnum, Option as ViewModeOptionEnum) as Variant

Specifies options for the control's view mode.

Type	Description
Mode as ViewModeEnum	A ViewModeEnum expression that indicates the way how data is displayed.
Option as ViewModeOptionEnum	A ViewModeOptionEnum expression that indicates the view's option being changed
Variant	A Variant expression indicates the newly value for the specified option.

The ViewModeOption property gets or sets a specified option for a particular view. Use the ViewMode option to change the way how data is displayed in the control. Use the BeginUpdate and EndUpdate methods to avoid painting the control while changing multiple options.

For instance, the following VB sample changes the size of the control's borders when the control is running in exTableView mode:

```
With Grid1
  .ViewModeOption(exTableView, exBorderWidth) = 2
  .ViewModeOption(exTableView, exBorderHeight) = 2
End With
```

property Grid.VirtualMode as Boolean

Specifies a value that indicates whether the control is running in the virtual mode.

Type	Description
Boolean	A boolean expression that indicates whether the control is running in the virtual mode.

Generally, the user needs to run the control in virtual mode, if a table with large number of records needs to be displayed. In virtual mode, the control handles maximum 2,147,483,647 records. **The control is running in virtual mode, only if the VirtualMode property is True, and the UnboundHandler property refers an object that implements the IUnboundHandler interface**. Implementing the IUnboundHandler interface is easy because it has only two methods. The first one, ItemsCount specifies the number of records that user needs to display in the control. The second method is ReadItem and it provides data for a specific record. When control is running in the virtual mode, the control loads **only** the items that need to be displayed. If the control is running in the unbound mode (the VirtualMode property is False), the control allocates memory for all records that need to be loaded. The data for each record is loaded only when it is required. The virtual mode has few disadvantages like: the sorting is not available (the user needs to provide sorting data), the control's filtering items is not available, the data cannot be viewed as a hierarchy, the user cannot add items manually, selection is not available, and so on. The main advantage of the virtual mode is that the control can displays large number of records. The unbound mode requires a lot of memory, depending on number of loaded records, but it allows almost all features of the control, including sorting, filtering and so on.

Use the ItemToVirtual property to convert the handle the item to the index of the virtual item. Use the VirtualToItem property to get the handle of the item giving the index of the virtual item. It is important to know, that the Items.VirtualToItem property ensures that the virtual item fits the control's client area, so calling the EnsureVisibleItem method is not required in this case. While running the control in virtual mode, you can use the Selection property to specify/retrieve the control's selection, or the index-es of the virtual selected items.

- Displaying a table, using the virtual mode (VB sample)
- Editing a table, using the virtual mode (VB sample)
- Adding a custom column, when the control is running in the virtual mode (VB sample)
- Loading and editing a table using virtual mode in C++ (VC++ sample)
- Running the VirtualMode in VFP 7.0 or greater (VFP sample)

Displaying a table, using the virtual mode

When you need to display large number of records, you need to provide an object that implements the IUnboundHandler interface. The object provides the number of records that needs to be displayed, and data for each record. The VirtualMode property needs to be set on true, and the object you have written needs to be passed to the UnboundHandler property.

The following VB sample adds a column, and 100 records. The index of each item is displayed.

- Create a new project (Project1)
- Add a control to the form (Grid1)
- Create a new class module (Class1) and add it to the project
- Open the code of the class, and type "Implements IUnboundHandler"
- Add the handler for the IUnboundHandler_ItemsCount property like follows:

```
Private Property Get IUnboundHandler_ItemsCount(ByVal Source As Object) As Long
   IUnboundHandler_ItemsCount = 100
End Property
```

The control calls the IUnboundHandler_ItemsCount property when the UnboundHandler property is set, to update the vertical scroll range.

- Add the handler for the IUnboundHandler_ReadItem method like follows:

```
Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As Object, ByVal ItemHandle As Long)
   With Source.Items
      .CellValue(ItemHandle, 0) = Index + 1
   End With
End Sub
```

The control calls the IUnboundHandler_ReadItem method each time when a virtual item becomes visible.

- Open the form's code and add handler for the Form_Load event like follows:

```
Private Sub Form_Load()
   With Grid1
      .BeginUpdate
         .Columns.Add "Column 1"
```

```
        .VirtualMode = True
        Set .UnboundHandler = New Class1
    .EndUpdate
  End With
End Sub
```

- Save the project
- Run the project

The sample runs the control in the virtual mode. The control calls the IUnboundHandler_ItemsCount property when UnboundHandler property is set. The IUnboundHandler_ReadItem method is invoked when a record needs to be displayed.

Now, that you got the idea of the virtual mode, let's start to complicate the things. Let's suppose that we have a table and we need to display its records in the control.

- Create a new project (Project1)
- Add a control to the form (Grid1)
- Create a new class module (Class1) and add it to the project
- Add a new variable rs, of Object type like: Public rs as Object. In the following sample, the rs variable holds a reference to an ADO.Recordset object
- Add a new procedure AttachTable like follows:

```
Public Sub AttachTable(ByVal strTable As String, ByVal strPath As String, ByVal g
As EXGRIDLibCtl.Grid)
  Set rs = CreateObject("ADODB.Recordset")
  rs.Open strTable, "Provider=Microsoft.Jet.OLEDB.4.0;Data Source= " & strPath,
3, 3
  With g
    .BeginUpdate
      With .Columns
        Dim f As Variant
        For Each f In rs.Fields
          .Add f.Name
        Next
      End With
    .EndUpdate
  End With
End Sub
```

The AttachTable subroutine opens a table using ADO, and insert in the control's Columns collection a new column for each field found in the table.

- Type "Implements IUnboundHandler" at the beginning of the class
- Implement the IUnboundHandler_ItemsCount property like follows:

```
Private Property Get IUnboundHandler_ItemsCount(ByVal Source As Object) As Long
   IUnboundHandler_ItemsCount = rs.RecordCount
End Property
```

In this case the IUnboundHandler_ItemsCount property the number of records in the table.

- Implement the IUnboundHandler_ReadItem method like follows:

```
Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As Object, ByVal ItemHandle As Long)
   rs.Move Index, 1
   Dim i As Long
   i = 0
   With Source.Items
      Dim f As Variant
      For Each f In rs.Fields
         .CellValue(ItemHandle, i) = f.Value
         i = i + 1
      Next
   End With
End Sub
```

The IUnboundHandler_ReadItem method moves the current record using the rs.Move method, at the record with the specified index, and loads values for each cell n the item. If you need to apply colors, font attributes, ... to the items in the control, your handler may change the CellBold, CellForeColor, ... properties like follows:

- Open the form's code, and add a new variable n like: Dim n As New Class1
- Add a handler for the Form_Load event like follows:

```
Private Sub Form_Load()
   With Grid1
      .BeginUpdate
```

```
        n.AttachTable "Select * from Orders",
"D:\Exontrol\ExGrid\sample\sample.mdb", Grid1

        .VirtualMode = True
        Set .UnboundHandler = n

    .EndUpdate
  End With
End Sub
```

The AttachTable method opens the table, and fills the control's Columns collection. The AttachTable method needs to be called before putting the control on virtual mode, because properties of the rs object are called in the ItemsCount and ReadItem methods.

- Save the project
- Run the project

Editing a table, using the virtual mode

In this case, we assume that you are already familiar with the "displaying a table, using virtual mode". So, beside the steps that need to be followed in "displaying a table, using virtual mode", the following steps need to be follow as well:

- Add editors for each column that require being editable like follows:

```
With .Columns("OrderDate")
  With .Editor
    .EditType = DateType
  End With
End With
```

- The Form_Load event should look like follows:

```
Private Sub Form_Load()
  With Grid1
    .BeginUpdate

        n.AttachTable "Select * from Orders",
```

```
"D:\Exontrol\ExGrid\sample\sample.mdb", Grid1

        .VirtualMode = True
        Set .UnboundHandler = n

          With .Columns("OrderDate")
             With .Editor
                .EditType = DateType
             End With
          End With

      .EndUpdate
   End With
End Sub
```

Important to notice is that setting editors is called after setting the UnboundHandler property. Also, the "OrderDate" field needs to be changed if another table or database is used. Until now, the sample is able to display the table, and it provides editors for the columns. Until now, the user can change the values in the control but the data is not saved to the table so please follow the steps:

- Handle the control's Change event like follows:

```
Private Sub Grid1_Change(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As
Long, newValue As Variant)
   With Grid1.Items
      n.Change .ItemToVirtual(Item), ColIndex, newValue
   End With
End Sub
```

The Change event passes the NewValue to the object that implements the IUnboundHandler interface, so we can make the change to the original place, in our case the recordset.

- The Change event is fired when user changes a value in the control. The Change event is called even is the user changes the cell's value using the CellValue property, so the IUnboundHandler_ReadItem needs a change like follows:

```
Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As
Object, ByVal ItemHandle As Long)
```

```
    nReading = nReading + 1
    rs.Move Index, 1
    Dim i As Long
    i = 0
    With Source.Items
        Dim f As Variant
        For Each f In rs.Fields
            .CellValue(ItemHandle, i) = f.Value
            i = i + 1
        Next
    End With
    nReading = nReading - 1
End Sub
```

Where is the change? The change is that we have added a counter nReading that is increases when the IUnboundHandler_ReadItem method starts and it is decreased when the function ends. Why such of counter? We have added the nReading counter because, during the IUnboundHandler_ReadItem method the user calls CellValue, so the Change event is fired and things get recursively as we do not want...

- Add a new method to the Class1 object like follows (Change):

```
Public Sub Change(ByVal Index As Long, ByVal ColIndex As Long, ByVal newValue
As Variant)
    If nReading = 0 Then
        rs.Move Index, 1
        rs(ColIndex) = newValue
    End If
End Sub
```

Checking the nReading counter is required because the Change event is called even if the user changes the cell's value using CellValue property. If such of checking is omitted, a recursive call occurs. The nReading counter is increased when the IUnboundHandler_ReadItem method starts, and the nReading counter is decreased when the IUnboundHandler_ReadItem method ends.

- The last thing that we need to add is to declare the variable (counter) nReading as Long: Dim nReading As Long, and to initialize it in the Class1 constructor like follows:

```
Private Sub Class_Initialize()
```

```
    nReading = 0
End Sub
```

- Save and run the project

Adding a custom column, when the control is running in the virtual mode.

Let's suppose that we want to display a column with the current position for each record in the table. In this case, we need to add a new column, and we need to change the ReadItem method like follows:

- The Form_Load event should look like:

```
Private Sub Form_Load()
    With Grid1
        .BeginUpdate

            n.AttachTable "Select * from Orders",
"D:\Exontrol\ExGrid\sample\sample.mdb", Grid1

            .VirtualMode = True
            Set .UnboundHandler = n

            With .Columns("OrderDate")
                With .Editor
                    .EditType = DateType
                End With
            End With

            With .Columns.Add("Position")
                .Position = 0
            End With

        .EndUpdate
    End With
End Sub
```

- The IUnboundHandler_ReadItem method looks like following:

```
Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As
Object, ByVal ItemHandle As Long)
   nReading = nReading + 1
   rs.Move Index, 1
   Dim i As Long
   i = 0
   With Source.Items
      Dim f As Variant
      For Each f In rs.Fields
         .CellValue(ItemHandle, i) = f.Value
         i = i + 1
      Next
       .CellValue(ItemHandle, "Position") = Index + 1
   End With
   nReading = nReading - 1
End Sub
```

For instance, if you need to have a column that computes its value based on the other columns, it can be done like this:

```
.CellValue(ItemHandle, "Column") = .CellValue(ItemHandle, "Quantity") *
.CellValue(ItemHandle, "UnitPrice")
```

Loading and editing a table using virtual mode in C++

The following tutorial will show how to run the control in virtual mode. The sample is a simple MFC dialog based application. Anyway, if your application is different than a MFC dialog based, the base things you need are here, so please find that the following information are useful.

- Create a new project using MFC AppWizard (exe) (ADOVirtual)
- Select Dialog based, for the type of the application
- Insert the control to the application's main dialog (Insert ActiveX Control)
- Save the Project
- Open the MFC Class Wizard, by pressing CTRL + W
- Add a new member variable for IDC_GRID1 resource called m_grid. In the meanwhile, please notice that the wizard will ask you 'The ActiveX Control "ExGrid ActiveX Control" has not been inserted into the project. Developer Studio will do this now and generate a C++ wrapper class for it', and you need to click ok, by following the steps that wizard will ask you to do in order to insert the C++ wrapper classese. (CGrid, CItems,

CColumn, CEditor, COleFont, CPicture, CColumns)

- Save the Project
- Open the Dialog Properties, and click the "Clip siblings" and "Clip children"
- Add a new MFC based class, CUnboundHandler derived from the CCmdTarget. We define the CUnboundHandler class to implement the IUnboundHandler interface.
- Import the control's definition using the #import directive like follows:

```
#import "c:\winnt\system32\exgrid.dll" rename( "GetItems", "exGetItems" )
```

The #import directive is used to incorporate information from a type library. The content of the type library is converted into C++ classes, mostly describing the COM interfaces. The path to the file need to be changed if the dll is somewhere else. After building the project, the environment generates a namespace EXGRIDLib. The generated namespace includes definition for IUnboundHandler interface. It can be accessed using the declaration EXGRIDLib::IUnboundHandler

- By default, the destructor of the CUnboundHandler class is declared as protected. The destructor needs to be declared as public (Remove the protected keyword before ~CUnboundHandler).
- Implementing the IUnboundHandler interface using the DECLARE_INTERFACE_MAP, BEGIN_INTERFACE_PART and END_INTERFACE_PART macros. The following snippet needs to be inserted in the class definition like

```
DECLARE_INTERFACE_MAP()

public:
   BEGIN_INTERFACE_PART(Handler, EXGRIDLib::IUnboundHandler)
         STDMETHOD(get_ItemsCount)(IDispatch * Source,long* pVal);
      STDMETHOD(raw_ReadItem)(long Index, IDispatch * Source, long
ItemHandle);
      END_INTERFACE_PART(Handler)
```

The CUnboundHandler class definition should look like follows (we have removed the comments added by the wizard):

```
#import "c:\winnt\system32\exgrid.dll" rename( "GetItems", "exGetItems" )
```

```
class CUnboundHandler : public CCmdTarget
{
   DECLARE_DYNCREATE(CUnboundHandler)
```

```
    CUnboundHandler();          // protected constructor used by dynamic
creation

    DECLARE_INTERFACE_MAP()

public:
    BEGIN_INTERFACE_PART(Handler, EXGRIDLib::IUnboundHandler)
        STDMETHOD(get_ItemsCount)(IDispatch * Source, long* pVal);
        STDMETHOD(raw_ReadItem)(long Index, IDispatch * Source, long
ItemHandle);
    END_INTERFACE_PART(Handler)

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CUnboundHandler)
    //}}AFX_VIRTUAL

// Implementation
    virtual ~CUnboundHandler();

    // Generated message map functions
    //{{AFX_MSG(CUnboundHandler)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()
};
```

- Add INTERFACE_PART definition in the UnboundHandler.cpp file like follows:

```
BEGIN_INTERFACE_MAP(CUnboundHandler, CCmdTarget)
    INTERFACE_PART(CUnboundHandler, __uuidof(EXGRIDLib::IUnboundHandler),
Handler)
END_INTERFACE_MAP()
```

- Write the get_ItemsCount property like follows:

```
STDMETHODIMP CUnboundHandler::XHandler::get_ItemsCount(IDispatch *
```

```
Source, long* pVal)
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    if ( pVal )
    {
        *pVal = 25000;
        return S_OK;;
    }
    return E_POINTER;
}
```

- Write the raw_ReadItem method like follows:

```
STDMETHODIMP CUnboundHandler::XHandler::raw_ReadItem(long Index,
IDispatch * Source, long ItemHandle)
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);

    // gets the source control
    EXGRIDLib::IGrid* pGrid = NULL;
    if ( SUCCEEDED( Source->QueryInterface( __uuidof(EXGRIDLib::IGrid),
(LPVOID*)&pGrid ) ) )
    {
        // assigns the value for each cell.
        pGrid->Items->CellValue[ItemHandle][_variant_t( (long)0 )] = _variant_t(
Index );
        pGrid->Release();
    }
    return S_OK;
}
```

- Add implementation for QueryInterface, AddRef and Release methods of IUnknown interface like follows:

```
STDMETHODIMP CUnboundHandler::XHandler::QueryInterface( REFIID riid,
void** ppvObject)
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
```

```
    if ( ppvObject )
    {
        if ( IsEqualIID( __uuidof(IUnknown), riid ) )
        {
            *ppvObject = static_cast<IUnknown*>( this );
            AddRef();
            return S_OK;
        }
        if ( IsEqualIID( __uuidof( EXGRIDLib::IUnboundHandler), riid ) )
        {
            *ppvObject = static_cast<EXGRIDLib::IUnboundHandler*>( this );
            AddRef();
            return S_OK;
        }
        return E_NOINTERFACE;
    }
    return E_POINTER;
}

STDMETHODIMP_(ULONG) CUnboundHandler::XHandler::AddRef()
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    return 1;
}

STDMETHODIMP_(ULONG) CUnboundHandler::XHandler::Release()
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    return 0;
}
```

- The CUnboundHandler class implementation should look like:

```
IMPLEMENT_DYNCREATE(CUnboundHandler, CCmdTarget)

BEGIN_INTERFACE_MAP(CUnboundHandler, CCmdTarget)
    INTERFACE_PART(CUnboundHandler, __uuidof(EXGRIDLib::IUnboundHandler),
```

```
Handler)
END_INTERFACE_MAP()

CUnboundHandler::CUnboundHandler()
{
}

CUnboundHandler::~CUnboundHandler()
{
}

STDMETHODIMP CUnboundHandler::XHandler::get_ItemsCount(IDispatch *
Source, long* pVal)
{
   METHOD_PROLOGUE(CUnboundHandler, Handler);
   if ( pVal )
   {
      *pVal = 25000;
      return S_OK;;
   }
   return E_POINTER;
}

STDMETHODIMP CUnboundHandler::XHandler::raw_ReadItem(long Index,
IDispatch * Source, long ItemHandle)
{
   METHOD_PROLOGUE(CUnboundHandler, Handler);

   // gets the source control
   EXGRIDLib::IGrid* pGrid = NULL;
   if ( SUCCEEDED( Source->QueryInterface( __uuidof(EXGRIDLib::IGrid),
(LPVOID*)&pGrid ) ) )
   {
      // assigns the value for each cell.
      pGrid->Items->CellValue[ItemHandle][_variant_t( (long)0 )] = _variant_t(
Index );
      pGrid->Release();
```

```
    }
    return S_OK;
}

STDMETHODIMP CUnboundHandler::XHandler::QueryInterface( REFIID riid,
void** ppvObject)
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    if ( ppvObject )
    {
        if ( IsEqualIID( __uuidof(IUnknown), riid ) )
        {
            *ppvObject = static_cast<IUnknown*>( this );
            AddRef();
            return S_OK;
        }
        if ( IsEqualIID( __uuidof( EXGRIDLib::IUnboundHandler), riid ) )
        {
            *ppvObject = static_cast<EXGRIDLib::IUnboundHandler*>( this );
            AddRef();
            return S_OK;
        }
        return E_NOINTERFACE;
    }
    return E_POINTER;
}

STDMETHODIMP_(ULONG) CUnboundHandler::XHandler::AddRef()
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    return 1;
}

STDMETHODIMP_(ULONG) CUnboundHandler::XHandler::Release()
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    return 0;
```

```
}

BEGIN_MESSAGE_MAP(CUnboundHandler, CCmdTarget)
   //{{AFX_MSG_MAP(CUnboundHandler)
      // NOTE - the ClassWizard will add and remove mapping macros here.
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

After all these steps we have defined the class CUnboundHandler that implements the IUnboundHandler interface. All that we need to do from now, is to add a column to the control, and to set the VirtualMode and UnboundHanlder properties like follows:

- Open the definition of the application's main dialog (CADOVirtualDlg)
- Include the definition of the CUnboundHandler class to CADOVirtualDlg using:

```
#include "UnboundHandler.h"
```

- Add a new member of CUnboundHandler type to the CADOVirtualDlg class like:

```
CUnboundHandler m_unboundHandler;
```

- Open the implementation file for the application's main dialog (CADOVirtualDlg)
- Add the definition for CColumns class (a wrapper class for the control) at the beginning of the file

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

- Locate the OnInitDialog() method and add the following code (after the "// TODO: Add extra initialization here"):

```
m_grid.BeginUpdate();
   m_grid.GetColumns().Add( _T("Column 1") );
   m_grid.SetVirtualMode( TRUE );
   m_grid.SetUnboundHandler( &m_unboundHandler.m_xHandler );
m_grid.EndUpdate();
```

- Save, Compile and Run the project

The tutorial shows how to put the control on virtual mode. The sample loads the numbers from 0 to 24999.

Now, that we got the idea how to implement the IUnboundHandler let's say that we want to

change the sample to load an edit an ADO recordset. The following tutorials shows how to display a table and how to add code in order to let user edits the data.

- Open the definition of the CUnboundHandler class
- Import the Microsoft ADO Type Library to the CUnboundHandler class like follows:

```
#import <msado15.dll> rename ( "EOF", "adoEOF" )
```

The #import directive generates the ADODB namspace. The ADODB namspace includes all definitions in the Microsoft ADO Type Library.

- Include a member of ADODB::_RecordsetPtr called m_spRecordset. The m_spRecordset member will handle data in the ADO table.

```
ADODB::_RecordsetPtr m_spRecordset;
```

- Add definition for AttachTable function like follows:

```
virtual void AttachTable( EXGRIDLib::IGrid* pGrid, LPCTSTR szTable, LPCTSTR
szDatabase );
```

Now, the CUnboundHandler class definition should look like follows:

```
#import "c:\winnt\system32\exgrid.dll" rename( "GetItems", "exGetItems" )
#import <msado15.dll> rename ( "EOF", "adoEOF" )

class CUnboundHandler : public CCmdTarget
{
   DECLARE_DYNCREATE(CUnboundHandler)

   CUnboundHandler();          // protected constructor used by dynamic creation

   DECLARE_INTERFACE_MAP()

public:
   BEGIN_INTERFACE_PART(Handler, EXGRIDLib::IUnboundHandler)
      STDMETHOD(get_ItemsCount)(IDispatch * Source, long* pVal);
      STDMETHOD(raw_ReadItem)(long Index, IDispatch * Source, long ItemHandle);
   END_INTERFACE_PART(Handler)
```

```
    virtual void AttachTable( EXGRIDLib::IGrid* pGrid, LPCTSTR szTable, LPCTSTR
szDatabase );

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CUnboundHandler)
    //}}AFX_VIRTUAL

// Implementation
    virtual ~CUnboundHandler();

    // Generated message map functions
    //{{AFX_MSG(CUnboundHandler)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()

    ADODB::_RecordsetPtr m_spRecordset;
};
```

- Open the implementation file for CUnboundHandler class (UnboundHandler.cpp file)
- Add the implementation for AttachTable function like follows:

```
void CUnboundHandler::AttachTable( EXGRIDLib::IGrid* pGrid, LPCTSTR szTable,
LPCTSTR szDatabase )
{
    if ( SUCCEEDED( m_spRecordset.CreateInstance( "ADODB.Recordset") ) )
    {
        try
        {
            CString strConnection = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=";
            strConnection += szDatabase;
            if ( SUCCEEDED( m_spRecordset->Open(_variant_t( szTable ),
_variant_t(strConnection), ADODB::adOpenStatic, ADODB::adLockPessimistic,
NULL ) ) )
```

```
        {
            pGrid->BeginUpdate();
            for ( long i = 0; i < m_spRecordset->Fields->GetCount(); i++ )
                pGrid->GetColumns()->Add( m_spRecordset->Fields->GetItem(
_variant_t( i ) )->Name );
            pGrid->EndUpdate();
        }
    }
    catch ( _com_error& e )
    {
        AfxMessageBox( e.Description() );
    }

  }
}
```

The AttachTable function opens a recordset, and adds a new column to the control's Columns collection for each field found in the recordset.

- Change the get_ItemsCount property like follows:

```
STDMETHODIMP CUnboundHandler::XHandler::get_ItemsCount(IDispatch *
Source, long* pVal)
{
  METHOD_PROLOGUE(CUnboundHandler, Handler);
  if ( pVal )
  {
    *pVal = pThis->m_spRecordset->RecordCount;
    return S_OK;;
  }
  return E_POINTER;
}
```

The ItemsCount property specifies that the control displays all records in the recordset

- Change the raw_ReadItem method like follows:

```
STDMETHODIMP CUnboundHandler::XHandler::raw_ReadItem(long Index,
IDispatch * Source, long ItemHandle)
```

```
{
   METHOD_PROLOGUE(CUnboundHandler, Handler);
   pThis->m_spRecordset->Move( Index, _variant_t(
(long)ADODB::adBookmarkFirst ) );

   // gets the source control
   EXGRIDLib::IGrid* pGrid = NULL;
   if ( SUCCEEDED( Source->QueryInterface( __uuidof(EXGRIDLib::IGrid),
(LPVOID*)&pGrid ) ) )
   {
      // assigns the value for each cell.
      for ( long i = 0; i < pThis->m_spRecordset->Fields->GetCount(); i++ )
         pGrid->Items->CellValue[ _variant_t( ItemHandle ) ][ _variant_t( i )] =
pThis->m_spRecordset->Fields->GetItem( _variant_t( i ) )->Value;
      pGrid->Release();
   }
   return S_OK;
}
```

The ReadItem method moves the position of the current record in the recordset, and sets the value for each cell in the item.

The implementation for CUnbundHandler class should look like:

```
IMPLEMENT_DYNCREATE(CUnboundHandler, CCmdTarget)

BEGIN_INTERFACE_MAP(CUnboundHandler, CCmdTarget)
   INTERFACE_PART(CUnboundHandler, __uuidof(EXGRIDLib::IUnboundHandler),
Handler)
END_INTERFACE_MAP()

CUnboundHandler::CUnboundHandler()
{
}

CUnboundHandler::~CUnboundHandler()
{
}
```

```
STDMETHODIMP CUnboundHandler::XHandler::get_ItemsCount(IDispatch *
Source, long* pVal)
{
   METHOD_PROLOGUE(CUnboundHandler, Handler);
   if ( pVal )
   {
      *pVal = pThis->m_spRecordset->RecordCount;
      return S_OK;;
   }
   return E_POINTER;
}

STDMETHODIMP CUnboundHandler::XHandler::raw_ReadItem(long Index,
IDispatch * Source, long ItemHandle)
{
   METHOD_PROLOGUE(CUnboundHandler, Handler);
   pThis->m_spRecordset->Move( Index, _variant_t(
(long)ADODB::adBookmarkFirst ) );

   // gets the source control
   EXGRIDLib::IGrid* pGrid = NULL;
   if ( SUCCEEDED( Source->QueryInterface( __uuidof(EXGRIDLib::IGrid),
(LPVOID*)&pGrid ) ) )
   {
      // assigns the value for each cell.
       for ( long i = 0; i < pThis->m_spRecordset->Fields->GetCount(); i++ )
          pGrid->Items->CellValue[ _variant_t( ItemHandle ) ][ _variant_t( i )]
= pThis->m_spRecordset->Fields->GetItem( _variant_t( i ) )->Value;
      pGrid->Release();
   }
   return S_OK;
}

void CUnboundHandler::AttachTable( EXGRIDLib::IGrid* pGrid, LPCTSTR
szTable, LPCTSTR szDatabase )
{
```

```
    if ( SUCCEEDED( m_spRecordset.CreateInstance( "ADODB.Recordset") ) )
    {
        try
        {
            CString strConnection = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=";
            strConnection += szDatabase;
            if ( SUCCEEDED( m_spRecordset->Open(_variant_t( szTable ),
_variant_t(strConnection), ADODB::adOpenStatic,
ADODB::adLockPessimistic, NULL ) ) )
            {
                pGrid->BeginUpdate();
                for ( long i = 0; i < m_spRecordset->Fields->GetCount(); i++ )
                    pGrid->GetColumns()->Add( m_spRecordset->Fields-
>GetItem( _variant_t( i ) )->Name );
                pGrid->EndUpdate();
            }
        }
        catch ( _com_error& e )
        {
            AfxMessageBox( e.Description() );
        }

    }
}

STDMETHODIMP CUnboundHandler::XHandler::QueryInterface( REFIID riid,
void** ppvObject)
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    if ( ppvObject )
    {
        if ( IsEqualIID( __uuidof(IUnknown), riid ) )
        {
            *ppvObject = static_cast<IUnknown*>( this );
            AddRef();
            return S_OK;
```

```
        }
        if ( IsEqualIID( __uuidof( EXGRIDLib::IUnboundHandler), riid ) )
        {
            *ppvObject = static_cast<EXGRIDLib::IUnboundHandler*>( this );
            AddRef();
            return S_OK;
        }
        return E_NOINTERFACE;
    }
    return E_POINTER;
}

STDMETHODIMP_(ULONG) CUnboundHandler::XHandler::AddRef()
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    return 1;
}

STDMETHODIMP_(ULONG) CUnboundHandler::XHandler::Release()
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    return 0;
}

BEGIN_MESSAGE_MAP(CUnboundHandler, CCmdTarget)
    //{{AFX_MSG_MAP(CUnboundHandler)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

- Locate the OnInitDialog method in the implementation file of the application's main dialog (AdoVirtualDlg.cpp)
- Add the following code to the OnInitDialog method:

```
EXGRIDLib::IGridPtr spGrid = NULL;
m_grid.GetControlUnknown()->QueryInterface( &spGrid );
m_grid.BeginUpdate();
    m_unboundHandler.AttachTable( spGrid, _T("Orders"),
```

```
_T("D:\\Exontrol\\ExGrid\\sample\\sample.mdb") );
    m_grid.SetVirtualMode( TRUE );
    m_grid.SetUnboundHandler( &m_unboundHandler.m_xHandler );
m_grid.EndUpdate();
```

The AttachTable function is called before setting the UnboundHandler property. The AttachTable function opens a recordset giving the SQL phrase and the database. The AttachTable function loads also the control's Columns collection from the Fields collection of the recordset.

- Save, Compile and Run the project

After all these your control will be able to display a table using the virtual mode. Now, we need to add some changes in order to let user edits data in the control using the control's collection of editors.

- Include the definition for the CColumns, CColumn, CEditor, CItems classes in the application's main dialog (CAdoVirtualDlg)

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

- Locate the OnInitDialog method in the implementation file of the application's main dialog (AdoVirtualDlg.cpp)
- Add editors for the columns like following:

```
m_grid.GetColumns().GetItem( _variant_t(_T("OrderDate"))
).GetEditor().SetEditType( EXGRIDLib::DateType );
m_grid.GetColumns().GetItem( _variant_t(_T("RequiredDate"))
).GetEditor().SetEditType( EXGRIDLib::DateType );
m_grid.GetColumns().GetItem( _variant_t(_T("ShippedDate"))
).GetEditor().SetEditType( EXGRIDLib::DateType );
```

The sample includes editors of DateType to "OrderDate", "RequiredDate" and "ShippedDate" columns. If the editor requires adding items or requires more changes, you could save the editor object to a variable like in the following sample:

```
_variant_t vtMissing; vtMissing.vt = VT_ERROR;
CEditor editor = m_grid.GetColumns().GetItem( _variant_t(_T("EmployeeID"))
```

```
).GetEditor();
    editor.SetEditType( EXGRIDLib::DropDownListType );
    editor.AddItem( 1, _T("Nancy Davolio"), vtMissing );
    editor.AddItem( 2, _T("Fuller Andrew"), vtMissing );
    editor.AddItem( 3, _T("Jannet Leverling"), vtMissing );
    editor.AddItem( 4, _T("Margaret Peacock"), vtMissing );
    editor.AddItem( 5, _T("Marius Buchanan"), vtMissing );
    editor.AddItem( 6, _T("Michael Suyama"), vtMissing );
    editor.AddItem( 7, _T("Robert King"), vtMissing );
    editor.AddItem( 8, _T("Laura Callahan"), vtMissing );
    editor.AddItem( 9, _T("Anne Dodsworth"), vtMissing );
```

- Add a handler for the Change event of the control.

```
void CADOVirtualDlg::OnChangeGrid1(long Item, long ColIndex, VARIANT FAR*
NewValue)
{
    m_unboundHandler.Change( NewValue, m_grid.GetItems().GetItemToVirtual(
Item ), ColIndex );
}
```

- Open the definition file of the CUnboundHandler class (UnboundHandler.h file)
- Add a new member variable of long type as:

```
long m_nReading;
```

The Change event is called even if the user changes the cell's value using the Cellvalue property. Because in the ReadItem method we are using the Cellvalue, we need to increase the m_nReading counter when ReadItem method starts, and decreases it when the function ends. So, we will be able to avoid recursive calls.

- Add the definition of the Change function in the CUnboundHandler class:

```
virtual void Change( VARIANT* pvtNewValue, long Index, long ColIndex );
```

The CUnboundHandler class definition should look like:

```
#import "c:\winnt\system32\exgrid.dll" rename( "GetItems", "exGetItems" )
#import <msado15.dll> rename ( "EOF", "adoEOF" )
```

```
class CUnboundHandler : public CCmdTarget
{
    DECLARE_DYNCREATE(CUnboundHandler)

    CUnboundHandler();          // protected constructor used by dynamic creation

    DECLARE_INTERFACE_MAP()

public:
    BEGIN_INTERFACE_PART(Handler, EXGRIDLib::IUnboundHandler)
        STDMETHOD(get_ItemsCount)(IDispatch * Source, long* pVal);
        STDMETHOD(raw_ReadItem)(long Index, IDispatch * Source, long ItemHandle);
    END_INTERFACE_PART(Handler)

    virtual void AttachTable( EXGRIDLib::IGrid* pGrid, LPCTSTR szTable, LPCTSTR
szDatabase );
    virtual void Change( VARIANT* pvtNewValue, long Index, long ColIndex );

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CUnboundHandler)
    //}}AFX_VIRTUAL

// Implementation
    virtual ~CUnboundHandler();

    // Generated message map functions
    //{{AFX_MSG(CUnboundHandler)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()

    ADODB::_RecordsetPtr m_spRecordset;
    long m_nReading;
};
```

- Init the counter in the CUnboundHandler class constructor like follows:

```
CUnboundHandler::CUnboundHandler()
{
    m_nReading = 0;
}
```

- Change the raw_ReadItem method like follows:

```
STDMETHODIMP CUnboundHandler::XHandler::raw_ReadItem(long Index,
IDispatch * Source, long ItemHandle)
{
   METHOD_PROLOGUE(CUnboundHandler, Handler);
   pThis->m_nReading++;
   pThis->m_spRecordset->Move( Index, _variant_t(
(long)ADODB::adBookmarkFirst ) );

   // gets the source control
   EXGRIDLib::IGrid* pGrid = NULL;
   if ( SUCCEEDED( Source->QueryInterface( __uuidof(EXGRIDLib::IGrid),
(LPVOID*)&pGrid ) ) )
   {
      // assigns the value for each cell.
      for ( long i = 0; i < pThis->m_spRecordset->Fields->GetCount(); i++ )
         pGrid->Items->CellValue[ _variant_t( ItemHandle ) ][ _variant_t( i )] =
pThis->m_spRecordset->Fields->GetItem( _variant_t( i ) )->Value;
      pGrid->Release();
   }
   pThis->m_nReading--;
   return S_OK;
}
```

- Add implementation for the Change function like:

```
void CUnboundHandler::Change( VARIANT* pvtNewValue, long Index, long
ColIndex )
{
   if ( m_nReading == 0 )
```

```
    {
        m_spRecordset->Move( Index, _variant_t( (long)ADODB::adBookmarkFirst ) );
        m_spRecordset->Fields->GetItem( _variant_t( ColIndex ) )->Value =
*pvtNewValue;
        m_spRecordset->Update();
    }
}
```

The Change function moves the position of the current record in the recordset, and updates the table. The control automatically will reread the record in order to update the date in the cells of the item, after Change event is processed.

Finally, the implementation of the CUnboundHandler class looks like:

```
IMPLEMENT_DYNCREATE(CUnboundHandler, CCmdTarget)

BEGIN_INTERFACE_MAP(CUnboundHandler, CCmdTarget)
    INTERFACE_PART(CUnboundHandler, __uuidof(EXGRIDLib::IUnboundHandler),
Handler)
END_INTERFACE_MAP()

CUnboundHandler::CUnboundHandler()
{
    m_nReading = 0;
}

CUnboundHandler::~CUnboundHandler()
{
}

STDMETHODIMP CUnboundHandler::XHandler::get_ItemsCount(IDispatch *
Source, long* pVal)
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    if ( pVal )
    {
        *pVal = pThis->m_spRecordset->RecordCount;
        return S_OK;;
```

```
    }
    return E_POINTER;
}

STDMETHODIMP CUnboundHandler::XHandler::raw_ReadItem(long Index,
IDispatch * Source, long ItemHandle)
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    pThis->m_nReading++;
    pThis->m_spRecordset->Move( Index, _variant_t(
(long)ADODB::adBookmarkFirst ) );

    // gets the source control
    EXGRIDLib::IGrid* pGrid = NULL;
    if ( SUCCEEDED( Source->QueryInterface( __uuidof(EXGRIDLib::IGrid),
(LPVOID*)&pGrid ) ) )
    {
        // assigns the value for each cell.
        for ( long i = 0; i < pThis->m_spRecordset->Fields->GetCount(); i++ )
            pGrid->Items->CellValue[ _variant_t( ItemHandle ) ][ _variant_t( i )] =
pThis->m_spRecordset->Fields->GetItem( _variant_t( i ) )->Value;
        pGrid->Release();
    }
    pThis->m_nReading--;
    return S_OK;
}

void CUnboundHandler::AttachTable( EXGRIDLib::IGrid* pGrid, LPCTSTR szTable,
LPCTSTR szDatabase )
{
    if ( SUCCEEDED( m_spRecordset.CreateInstance( "ADODB.Recordset") ) )
    {
        try
        {
            CString strConnection = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=";
            strConnection += szDatabase;
```

```
        if ( SUCCEEDED( m_spRecordset->Open(_variant_t( szTable ),
_variant_t(strConnection), ADODB::adOpenStatic, ADODB::adLockPessimistic,
NULL ) ) )
        {
            pGrid->BeginUpdate();
            for ( long i = 0; i < m_spRecordset->Fields->GetCount(); i++ )
                pGrid->GetColumns()->Add( m_spRecordset->Fields->GetItem(
_variant_t( i ) )->Name );
            pGrid->EndUpdate();
        }
    }
    catch ( _com_error& e )
    {
        AfxMessageBox( e.Description() );
    }

  }
}

void CUnboundHandler::Change( VARIANT* pvtNewValue, long Index,
long ColIndex )
{
  if ( m_nReading == 0 )
  {
      m_spRecordset->Move( Index, _variant_t(
(long)ADODB::adBookmarkFirst ) );
      m_spRecordset->Fields->GetItem( _variant_t( ColIndex ) )->Value =
*pvtNewValue;
      m_spRecordset->Update();
  }
}

STDMETHODIMP CUnboundHandler::XHandler::QueryInterface( REFIID riid,
void** ppvObject)
{
  METHOD_PROLOGUE(CUnboundHandler, Handler);
  if ( ppvObject )
```

```
    {
        if ( IsEqualIID( __uuidof(IUnknown), riid ) )
        {
            *ppvObject = static_cast<IUnknown*>( this );
            AddRef();
            return S_OK;
        }
        if ( IsEqualIID( __uuidof( EXGRIDLib::IUnboundHandler), riid ) )
        {
            *ppvObject = static_cast<EXGRIDLib::IUnboundHandler*>( this );
            AddRef();
            return S_OK;
        }
        return E_NOINTERFACE;
    }
    return E_POINTER;
}

STDMETHODIMP_(ULONG) CUnboundHandler::XHandler::AddRef()
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    return 1;
}

STDMETHODIMP_(ULONG) CUnboundHandler::XHandler::Release()
{
    METHOD_PROLOGUE(CUnboundHandler, Handler);
    return 0;
}

BEGIN_MESSAGE_MAP(CUnboundHandler, CCmdTarget)
    //{{AFX_MSG_MAP(CUnboundHandler)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

If you need to apply colors, font attributes, … for items or cells while the control is running

in the virtual mode, the changes should be done in the raw_ReadItem method like follows:

```
STDMETHODIMP CUnboundHandler::XHandler::raw_ReadItem(long Index,
IDispatch * Source, long ItemHandle)
{
   METHOD_PROLOGUE(CUnboundHandler, Handler);
   pThis->m_nReading++;
   pThis->m_spRecordset->Move( Index, _variant_t(
(long)ADODB::adBookmarkFirst ) );

   // gets the source control
   EXGRIDLib::IGrid* pGrid = NULL;
   if ( SUCCEEDED( Source->QueryInterface( __uuidof(EXGRIDLib::IGrid),
(LPVOID*)&pGrid ) ) )
   {
      // assigns the value for each cell.
      for ( long i = 0; i < pThis->m_spRecordset->Fields->GetCount(); i++ )
         pGrid->Items->CellValue[ _variant_t( ItemHandle ) ][ _variant_t( i )] =
pThis->m_spRecordset->Fields->GetItem( _variant_t( i ) )->Value;

      if ( pGrid->Items->CellValue[ _variant_t( ItemHandle ) ][ _variant_t(
_T("ShipRegion") )] == _variant_t( _T("RJ") ) )
         pGrid->Items->put_ItemForeColor( ItemHandle , RGB(0,0,255 ) );
      if ( pGrid->Items->CellValue[ _variant_t( ItemHandle ) ][ _variant_t(
_T("ShipRegion") )] == _variant_t( _T("SP") ) )
         pGrid->Items->put_ItemBold( ItemHandle , TRUE );

      pGrid->Release();
   }
   pThis->m_nReading--;
   return S_OK;
}
```

While compiling the project the compiler displays warnings like: "warning C4146: unary minus operator applied to unsigned type, result still unsigned". You have to include the :

```
#pragma warning( disable : 4146 )
```

before importing the type libraries.

```
#pragma warning( disable : 4146 )
#import "c:\winnt\system32\exgrid.dll" rename( "GetItems", "exGetItems" )
#import <msado15.dll> rename ( "EOF", "adoEOF" )
```

Running the VirtualMode in VFP 7.0 or greater

1. Create a PRG file, say, CLASS1.PRG to define a class to implement the IUnboundHandler interface:

```
define class UnboundHandler as custom
   implements IUnboundHandler in "ExGrid.dll"

   function IUnboundHandler_get_ItemsCount(Source)
      return 10000000   && we are going to have that many virtual items
   endfunc

   function IUnboundHandler_ReadItem(Index, Source, ItemHandle)
      With Source.Items
      .DefaultItem = ItemHandle
            .CellValue(0, 0) = 'Virtual Item ' + transform(Index+1)    && in this example,
just set text in Column 0
      EndWith
   endfunc

enddefine
```

2. Set up ExGrid's properties as follows (in the example, it is done by Form1.Init)

```
with thisform.Grid1
   .Columns.Add("Column 0")
   .VirtualMode = .t.
   .UnboundHandler = newobject('UnboundHandler', 'class1.prg')
endwith
```

3. You can also use Virtual Mode to scan a table, say, MyCursor, in natural order.

```
function IUnboundHandler_get_ItemsCount(Source)
```

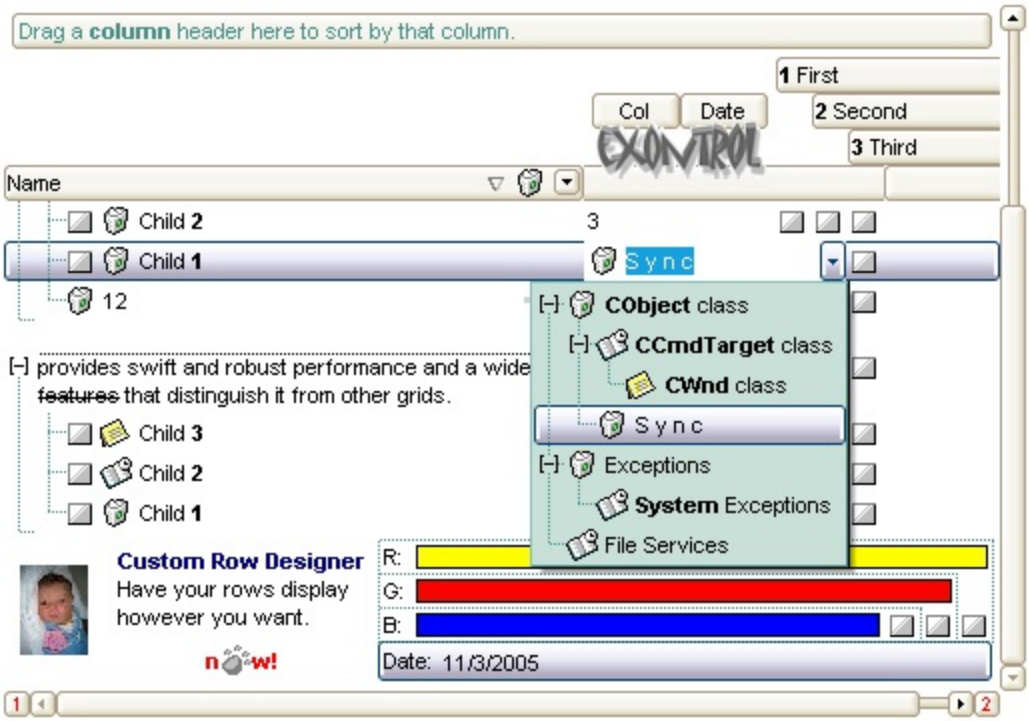
```
    return reccount('MyCursor')
endfunc

function IUnboundHandler_ReadItem(Index, Source, ItemHandle)
    select MyCursor
    go Index+1
    With Source.Items
        .DefaultItem = ItemHandle
            .CellValue(0, 0) = 'Virtual Item ' + MyCursor.Field1
    EndWith
endfunc
enddefine
```

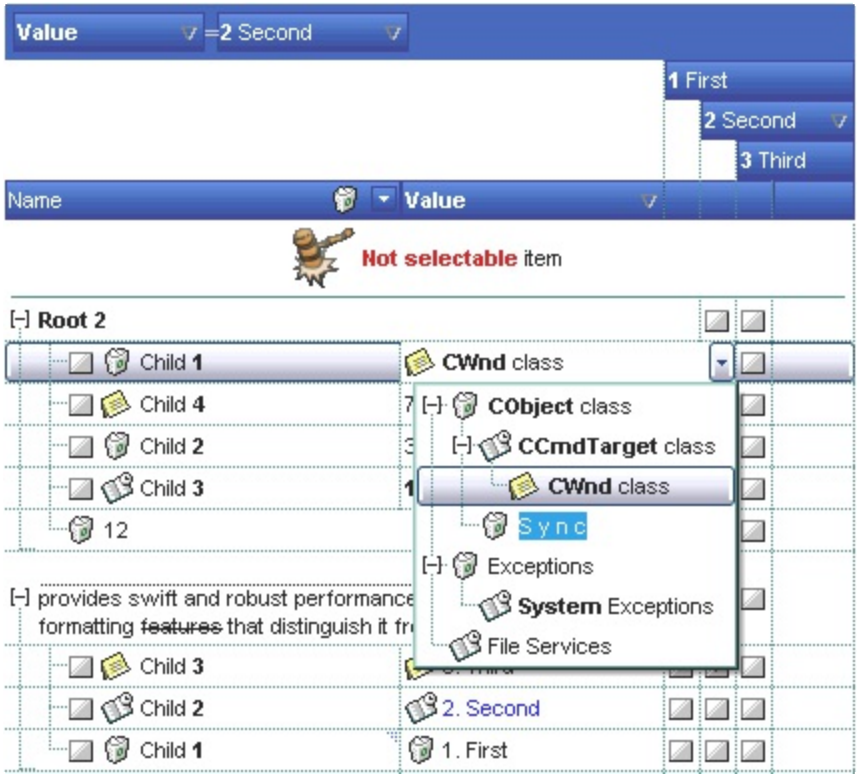
property Grid.VisualAppearance as Appearance

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 Grid.VisualDesign as String

Invokes the control's VisualAppearance designer.

Type	Description
String	A String expression that encodes the control's Visual Appearance.

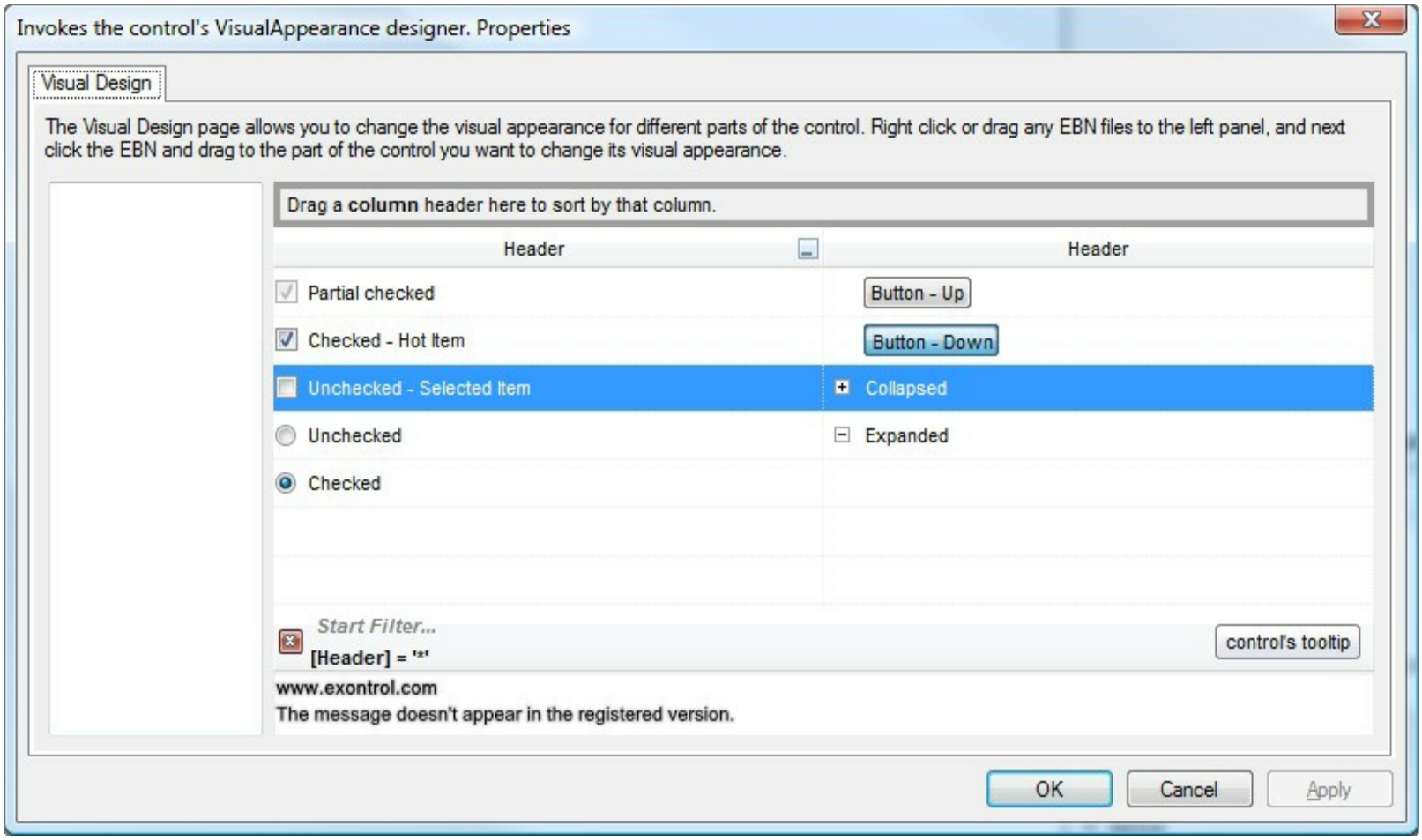
By default, the VisualDesign property is "". The VisualDesign property helps you to define fast and easy the control's visual appearance using the XP-Theme elements or EBN objects. The VisualDesign property can be accessed on design mode, and it can be used to design the visual appearance of different 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.

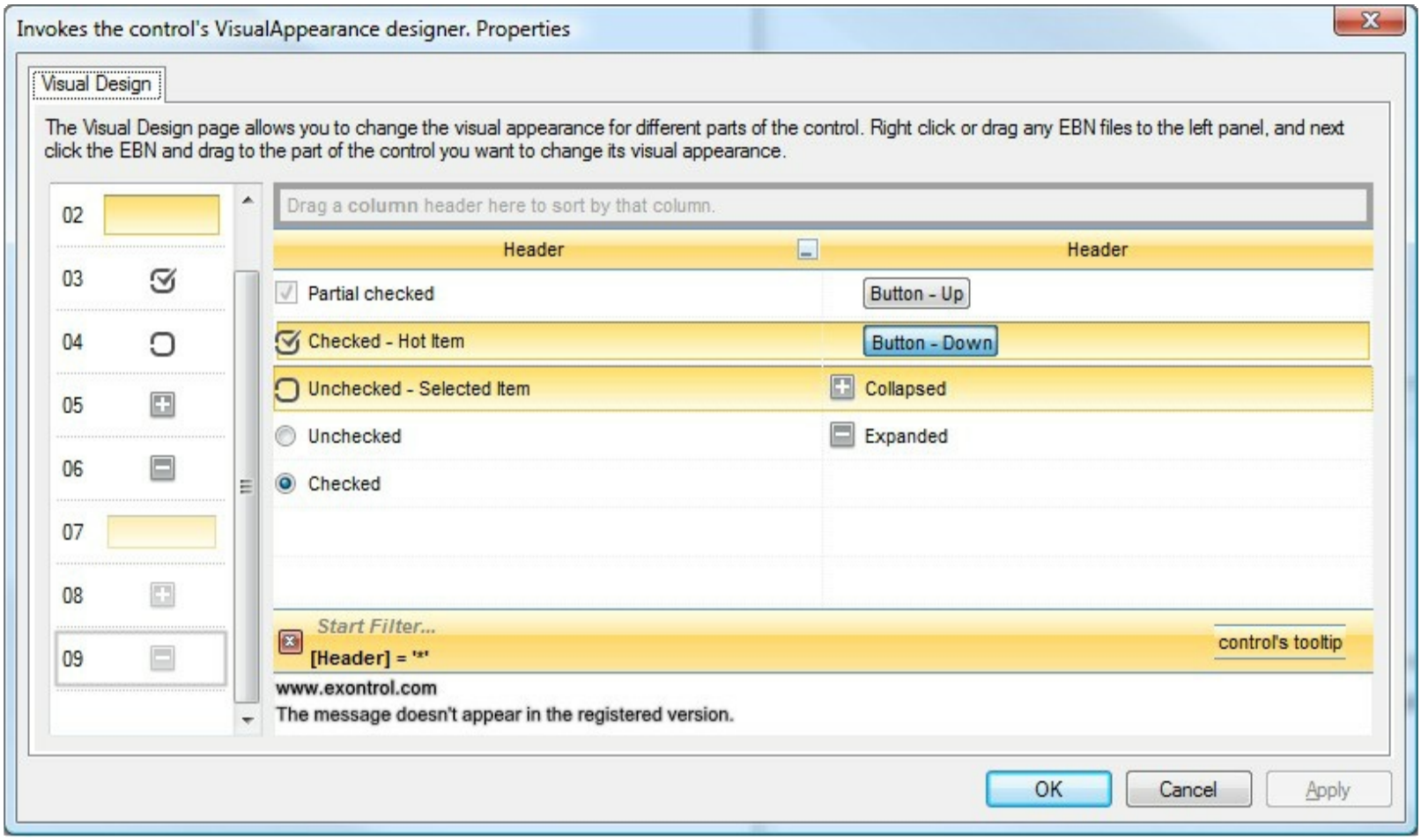
- Click here to watch a movie on how you define the control's visual appearance using the XP-Theme
- Click here to watch a movie on how you define the control's visual appearance using the EBN files.

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):



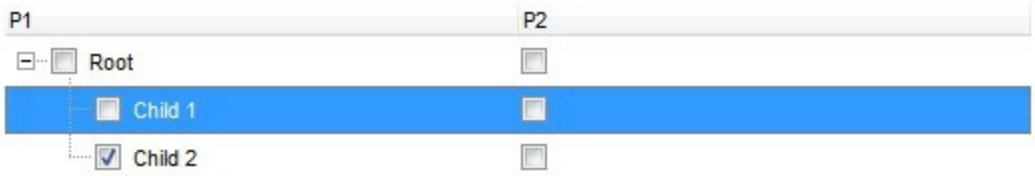
The following picture shows the control's VisualDesign form after applying some EBN objects:



This layout generates the following code:

```
With Exgrid1
   .VisualDesign =
"gBFLBWIgBAEHhEJAEGg7oB0HBSQAwABsIfj/jEJAcKhYEjgCAscA8ThQBA8cAgIjgDh8KBAPj
 & _

"RuF6FxmAkchiheZg5gYZIW0yMhZhqD55jIboamcCY2HGG5nCmVh0h2ZYUAyCQ4Xqbh9h8
 & _

"o5B8MwE4HsD4/g/ijHQHoLwrxUjrH0H4Z4rR2h7A8N8UggRNBnGCP8eA/A/gXGSPMfg3w
 & _

"DCDgJQFICxhDQGYBofYQYFCwD4J+XYQwIBECiCwJIExhhnCIDoNAnhzj8CyBcIosQ+BlAwM
 & _

"J8YQlwaBMCaCMd6hRnBpE+HoIwIQ9hdEKM8VYawoCcC8BUSYtxqBuDuFsOwTgLgLhZhAh
 & _

"zhGhtoEB+AsArhnhLhehUB5BfA4BfARBPgWB9h3hhBZB/AvA+BzhkhLhCh7hPg8g1BfhzAKE
 & _

"hQH1hSgAgcAmghgIg2AugLBigiBqAnAzBiVdgIA1ANAjBEgbAmAJMwA+gLgjgyBWA4A0E
 & _

"IAUgCA0AMhjA0ggWUgjh+GhBihI1yAKhiByBqAkV1gCAKAiV3141516g+Jmhj19V+V/AI2A

End With
```

If running the empty control we get the following picture:



If running the control using the code being generated by the VisualAppearance designer we get:

P1	P2
Root	
Child 1	
Child 2	

property Grid.WordFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS, Highlight as Boolean, [Reserved as Variant]) as String

Retrieves the word 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 X location of the mouse pointer. The x values is always expressed in client coordinates.
Highlight as Boolean	A Boolean expression that indicates whether the word from the position is highlighted.
Reserved as Variant	A long expression that specifies the part/parts of the control to search for the word from the cursor.
String	A String expression that indicates the word from the cursor.

Use the WordFromPoint property to retrieve the word from the cursor. 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 WordFromPoint property determines the index word from the cursor**. By default, the WordFromPoint property looks for the words in the Items area but it can search for words in any part of the control (Reserved parameter). Shortly, in the area where the items are displayed. A word is being defined as the sequence of the characters between two space/tab characters (empty characters). The word being returned does not include any HTML tags, in case the cursor hovers an HTML text. Use the ItemFromPoint property to get the item or cell from the cursor. Use the AnchorFromPoint property to determine the identifier of the anchor from the point. Use the CellCaption property to retrieve the entire cell's caption. Use the CellValue property to retrieve the cell's value.

The following VB sample displays the word from the cursor:

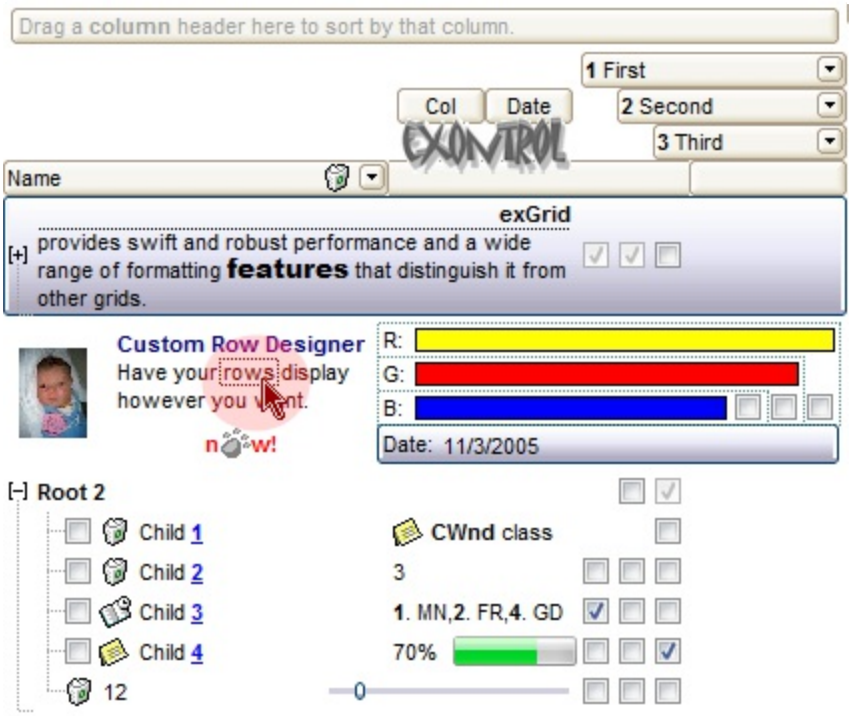
```
Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   Debug.Print Grid1.WordFromPoint(-1, -1)
End Sub
```

The following VB sample highligths the word from the cursor, as soon as the cursor hovers

a word:

```
Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Debug.Print Grid1.WordFromPoint(-1, -1, True)
End Sub
```

The following screen shot shows the "rows" word being highlighted when the cursor hovers it:



As soon as the cursor hovers another word it gets highlighted. The highlighting is temporary so as soon as the control is repainted the highlight is lost. For instance, you resize a column, scroll, or select a new item.

The following VB sample highlight the word from the cursor, and displays a context menu (eXPopupMenu) when the user right clicks the control:

```
Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Debug.Print Grid1.WordFromPoint(-1, -1, True)
End Sub

Private Sub Grid1_RClick()
   Dim i As Long
   With PopupMenu1.Items
      .Clear
```

```
    .Add Grid1.WordFromPoint(-1, -1, True)
  End With
  i = PopupMenu1.ShowAtCursor()
End Sub
```

Currently, the Reserved parameter could be a combination (bitwise OR) of any of the following:

- (1) - Items area (the area where the items are shown)
- (2) - Header area (the area where the column's caption is displayed)
- (4) - SortBar area (the part of the control that shows the sorting columns, if multiple columns sorting is enabled)
- (8) - FilterBar area (the part of the control that displays the filter bar)

For instance, if the Reserved parameter is 1 + 2 (3), the WordFromPoint property retrieves the word from Items or Header area as well. If missing the control looks for the word in the Items section.

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	Verifies whether the item can be child of another item.
AddItem	Adds a new item, and returns a handle to the newly created item.
CellBackColor	Retrieves or sets the cell's background color.
CellBold	Retrieves or sets a value that specifies whether the cell should appear in bold.
CellButtonAutoWidth	Retrieves or sets a value indicating whether the cell's button fits the cell's caption.
CellCaption	Gets the cell's display value.
CellChecked	Retrieves the cell's handle that is checked giving the radio group identifier.
CellData	Specifies the cell's extra data.
CellEditor	Creates and gets the cell's built-in editor.
CellEditorVisible	Specifies whether column's editor is visible or hidden in the 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.
CellFormatLevel	Specifies the arrangement of the fields inside the cell.
CellHAlignment	Retrieves or sets a value that indicates the alignment of the cell's caption.
CellHasButton	Retrieves or sets a value indicating whether the cell has associated a push button.
CellHasCheckBox	Retrieves or sets a value indicating whether the cell has associated a checkbox.

CellHasRadioButton	Retrieves or sets a value indicating whether the cell has associated a radio button.
CellHyperLink	Specifies whether the cell's is highlighted when the cursor mouse is over the cell.
CellImage	Retrieves or sets a value that indicates the index of icon in the cell.
CellImages	Specifies an additional list of icons shown in the cell.
CellItalic	Retrieves or sets a value that specifies whether the cell should appear in italic.
CellItem	Retrieves the handle of item that is the owner of a specfic cell.
CellMerge	Retrieves or sets a value that indicates the index of the cell that's merged to.
CellOwnerDraw	Specifies the cell's owner draw handler.
CellParent	Retrieves the parent of an inner cell.
CellPicture	Retrieves or sets the cell's picture.
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.
CellSortData	Specifies the cell's sort data.
CellState	Retrieves or sets the cell's state. Has effect only for check and radio cells.
CellStrikeOut	Retrieves or sets a value that specifies whether the cell should appear in strikeout.
CellToolTip	Retrieves or sets a value that indicates the cell's too tip
CellUnderline	Retrieves or sets a value that specifies whether the cell should appear in underline.
CellVAlignment	Retrieves or sets a value that indicates how the cell's caption is vertically aligned.
CellValue	Specifies the cell's value.

CellValueFormat	Specifies how the cell's caption is displayed.
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.
CollapseAllCards	Collapses all the cards.
ComputeValue	Computes the value of a specified formula.
DefaultItem	Retrieves or sets a value that indicates the handle of the item used by Items properties in VFP.
DeleteCellEditor	Deletes the cell's built-in editor created by CellEditor property.
EnableItem	Returns or sets a value that determines whether a item can respond to user-generated events.
EndBlockUndoRedo	Ends recording the UI operations and adds the undo/redo operations as a block, so they all can be restored at once, if Undo method is performed.
EnsureVisibleItem	Ensures the given item is in the visible client area.
ExpandAllCards	Expands all the cards.
ExpandCard	Expands or collapses the card.
ExpandItem	Expands, or collapses, the child items of the specified item.
FindItem	Finds an item, looking for Value 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 in the control.
FocusItem	Retrieves the handle of item that has the focus.
FormatCell	Specifies the custom format to display the cell's content.
	Returns the fully qualified path of the referenced item in

FullPath	the ExGrid control. The value is taken from the column SearchColumnIndex.
GroupItem	Indicates a group item if positive, and the value specifies the index of the column that has been grouped.
HasCellEditor	Specifies whether a cell has a built-in editor.
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.
InsertObjectItem	Inserts a new item that hosts the giving object, 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 given the item and the 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.

ItemFiltered	Checks whether the item is included in the control's filter.
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 the InsertControlItem method.
ItemParent	Returns the handle of the item's parent item.
ItemPosition	Retrieves or sets a value that indicates the item's position in the children list.
ItemStrikeOut	Retrieves or sets a value that indicates whether the item should appear in strikeout.
ItemToIndex	Retrieves the index of item in the Items collection given its handle.
ItemToVirtual	Gets the index of the virtual item giving the handle of the item.
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 the InsertControlItem method.
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 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 in the Items collection.
RootItem	Retrieves the handle of the root item giving its index in 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.
Selection	Selects items by index.
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 grid, only the immediate children of Item will be sorted.
SplitCell	Splits a cell, and returns the inner created cell.
StartBlockUndoRedo	Starts recording the UI operations as a block of undo/redo

	operations.
UnmergeCells	Unmerges a list of cells.
UnselectAll	Unselects all items.
UnsplitCell	Unsplits a cell.
VirtualToItem	Gets the handle of the item giving the index of the virtual item.
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

Verifies whether the item can be the child of another item

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

The AcceptSetParent property doesn't change the parent item. Use the SetParent method to change the item's parent. Use the ItemParent property to retrieve the item's parent. Use the InsertItem method to add child items to another item. An item is called root, if it has no parent (ItemParent() gets 0).

method Items.AddItem ([Value as Variant])

Adds a new item, and returns a handle to the newly created item.

Type	Description
Value as Variant	A variant expression that indicates the cell's value for the first column or a safe array that holds the values for each column.

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

Use the AddItem property to add new items/cards that have no parent (usually when your control acts like a list or in CardView mode). Adding new items fails, if the control has no columns. Use the Add method to add new columns to the control. Use InsertItem to insert child items (usually when your control acts like a tree). Use the InsertControlItem to insert ActiveX items. Use PutItems method to load an array of variants. When a new item is added to the Items collection, the control fires the AddItem event. If the control contains more than one column use the CellValue property to set the cell's value. Use the CellValueFormat property to specify whether the value contains HTML format or computed fields. If the control has no columns the AddItem method fails. Use Add method to insert new columns to the control. The AddItem method is not available if the control is running in the virtual mode. Use the LockedItemCount property to lock or unlock items to the top or bottom side of the control. Use the MergeCells method to combine one or more cells in a single cell. Use the SplitCell property to split a cell. If the CauseValidateValue property is True, the control fires the ValidateValue property when the user adds a new item. Please notice that the Change event is fired when adding a new item, if you are specifying a value using the Value argument). Use the BeginUpdate and EndUpdate methods to maintain performance while adding new columns and items. Use the CellEditor property to assign an editor to a cell. Use the Editor property to assign an editor to all cells in the column. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB6 sample uses the VB Array function to add two items:

```
With Grid1
  .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 Grid1
    .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:

```
exgrid1.BeginUpdate();

exgrid1.Columns.Add("Column 1");
exgrid1.Columns.Add("Column 2");
exgrid1.Columns.Add("Column 3");

exgrid1.Items.AddItem(new object[] { "Item 1.1", "Item 1.2", "Item 1.3" });
exgrid1.Items.AddItem(new object[] { "Item 2.1", "Item 2.2", "Item 2.3" });

exgrid1.EndUpdate();
```

The following C# sample adds a new item to the control:

```
EXGRIDLib.Items items = axGrid1.Items;
int i = items.AddItem("new items");
EXGRIDLib.Editor editor = items.get_CellEditor(i, 0);
editor.EditType = EXGRIDLib.EditTypeEnum.EditType;
```

The following VB.NET sample adds a new item to the control:

```
With AxGrid1.Items
   Dim i As Integer = .AddItem("new item")
   With .CellEditor(i, 0)
      .EditType = EXGRIDLib.EditTypeEnum.EditType
   End With
End With
```

The following C++ sample adds a new item to the control:

```
#include "Items.h"
#include "Editor.h"
COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
CItems items = m_grid.GetItems();
COleVariant vtItem( items.AddItem( COleVariant("new item") ) ), vtColumn( long(0) );
CEditor editor = items.GetCellEditor( vtItem, vtColumn );
editor.SetEditType( 1 /*EditType*/ );
```

The following VFP sample adds a new item to the control:

```
with thisform.Grid1.Items
   .DefaultItem = .AddItem("new item")
   With .CellEditor(0, 0)
      .EditType = 1 && EditType
   EndWith
endwith
```

The following VB sample uses the VB Array function to add items to a multiple columns control:

```
With Grid1
   .BeginUpdate
```

```
    .LinesAtRoot = exLinesAtRoot
    .HasLines = exNoLine
    .HasButtons = exArrow

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

    With .Items
        Dim h As HITEM
        h = .AddItem(Array("Cell 1", "Cell 2"))
        .InsertItem h, , Array("Sub Cell 1.1", "Sub Cell 2.1")
        .InsertItem h, , Array("Sub Cell 1.2", "Sub Cell 2.2")

    End With

    .EndUpdate
End With
```

The following VB sample adds items to the control:

```
' Adds two columns
With .Columns
    With .Add("Column 1").Editor
        .EditType = EditTypeEnum.ColorType
    End With
    With .Add("Column 2").Editor
        .EditType = EditTypeEnum.EditType
    End With
End With

'Adds few items
With .Items
    Dim h As HITEM
    h = .AddItem(vbRed)
    .CellValue(h, 1) = "Simple Text"
    h = .AddItem(vbBlue)
```

```
    .CellValue(h, 1) = "Simple Text"
End With
```

The following VB sample loads a table row by row:

```
' Creates an ADO Recordset
Set rs = CreateObject("ADODB.Recordset")
rs.Open "Employees", "Provider=Microsoft.Jet.OLEDB.4.0;Data Source= D:\Program
Files\Microsoft Visual Studio\VB98\NWIND.MDB", 3

With .Columns
    Dim f As Object
    For Each f In rs.Fields
        .Add f.Name
    Next
End With

' Adds an item for each record
With .Items
    rs.MoveFirst
    While Not rs.EOF
        .AddItem rs(0).Value
        rs.MoveNext
    Wend
End With
```

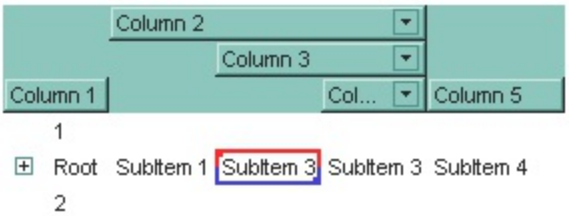
```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
    ' Gets the value for each field, in the current record
    With Grid1.Items
        .ItemData(Item) = rs.Bookmark
        For i = 1 To Grid1.Columns.Count - 1
            .CellValue(Item, i) = rs(i).Value
        Next
    End With
End Sub
```

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

Retrieves or sets the cell's background color.

Type	Description
Item as Variant	A long expression that indicates the item's handle
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key
Color	A color expression that indicates the cell's background color.

To change the background color for the entire item you can use ItemBackColor property. Use the ClearCellBackColor method to clear the cell's background color. Use the BackColor property to specify the control's background color. Use the CellForeColor property to specify the cell's foreground color. Use the ItemForeColor property to specify the item's foreground color. Use the 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 Grid1
   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_grid.GetVisualAppearance().Add( 0x40,
COleVariant(_T("D:\\Temp\\ExGrid.Help\\cell.ebn")) );
m_grid.GetItems().SetCellBackColor( COleVariant( m_grid.GetItems().GetFirstVisibleItem() ),
COleVariant( long(0) ), 0x40000000 );
```

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

```
With AxGrid1
   With .VisualAppearance
      .Add(&H40, "D:\Temp\ExGrid.Help\cell.ebn")
   End With
   With .Items
      .CellBackColor(.FirstVisibleItem, 0) = &H40000000
   End With
End With
```

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

```
axGrid1.VisualAppearance.Add(0x40, "D:\\Temp\\ExGrid.Help\\cell.ebn");
axGrid1.Items.set_CellBackColor(axGrid1.Items.FirstVisibleItem, 0, 0x40000000);
```

The following VFP sample changes the cell's appearance.

```
With thisform.Grid1
   With .VisualAppearance
      .Add(64, "D:\Temp\ExGrid.Help\cell.ebn")
   EndWith
   with .Items
      .DefaultItem = .FirstVisibleItem
      .CellBackColor(0,0) = 1073741824
   endwith
EndWith
```

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

You can use the following VB.NET function:

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

You can use the following C# function:

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

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

```
axGrid1.Items.set_CellBackColor(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex,
ToUInt32(Color.Red));
```

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

```
With AxGrid1.Items
   .CellBackColor(.FocusItem, AxGrid1.FocusColumnIndex) = ToUInt32(Color.Red)
End With
```

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

```
#include "Items.h"
CItems items = m_grid.GetItems();
items.SetCellBackColor( COleVariant( items.GetFocusItem() ), COleVariant(
m_grid.GetFocusColumnIndex() ), RGB(255,0,0) );
```

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

```
With Grid1.Items
   .CellBackColor(.FocusItem, Grid1.FocusColumnIndex) = vbRed
End With
```

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

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .CellBackColor( 0, thisform.Grid1.FocusColumnIndex ) = RGB(255,0,0)
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

Retrieves or sets a value that specifies whether the cell should appear in bold.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key
Boolean	A boolean expression that indicates whether the cell should appear in bold.

Use the CellBold property to bold a cell. Use the ItemBold property to specify whether the item should appear in bold. Use the HeaderBold property of the Column object to bold the column's caption. Use the CellItalic, CellUnderline or CellStrikeOut property to apply different font attributes to the cell. Use the ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellValueFormat 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 focused cell:

```
With Grid1.Items
   .CellBold(.FocusItem, Grid1.FocusColumnIndex) = True
End With
```

The following C++ sample bolds the focused cell:

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

The following C# sample bolds the focused cell:

```
axGrid1.Items.set_CellBold(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex , true);
```

The following VB.NET sample bolds the focused cell:

```
With AxGrid1.Items
   .CellBold(.FocusItem, AxGrid1.FocusColumnIndex) = True
End With
```

The following VFP sample bolds the focused cell:

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .CellBold( 0, thisform.Grid1.FocusColumnIndex ) = .t.
endwith
```

The following VB sample bolds all cells in the first column:

```
Dim h As Variant
Grid1.BeginUpdate
With Grid1.Items
For Each h In Grid1.Items
   .CellBold(h, 0) = True
Next
End With
Grid1.EndUpdate
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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 cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression 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 CellValue property to specify the button's caption. Use the CellValueFormat property to assign an HTML caption to the button. The control fires the ButtonClick property when the user clicks a button.

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

Gets the cell's display value.

Type	Description
Item as Variant	A long expression that indicates the item's handle. During the ValidateValue event, you can uses -1 instead Item, to access to the modified value. In other words during ValidateValue event, the Items.CellValue(Item,ColIndex) and Items.CellCaption(Item,ColIndex) properties retrieve the original value/caption of the cell while the Items.CellValue(-1,ColIndex) and Items.CellCaption(-1,ColIndex) gets the modified value of the specified cell.
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 value as it is displayed on the user interface.

The CellCaption property retrieves the cell's display value as it is displayed on the control's user interface. If the cell has no editor associated (no editor was assigned to the column and no editor was assigned to the cell), the CellCaption property gets the string representation of the cell's value. Use the CellValue property to change the cell's value. For instance, if a cell has a drop down list editor, the CellCaption property retrieves the caption of the predefined values. Use the CellImage property to assign an icon to a cell. Use the CellImages property to assign multiple icons to a cell. Use the CellPicture property to assign a custom size picture to a cell.

property Items.CellChecked (RadioGroup as Long) as HCELL

Retrieves the handle of the cell that is checked, given the radio group identifier.

Type	Description
RadioGroup as Long	A long expression that indicates the radio group identifier.
HCELL	A long expression that indicates the cell's handle. Use the CellItem property to retrieve the handle of the owner item.

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

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

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
   Grid1.Items.CellHasRadioButton(Item, 0) = True
   Grid1.Items.CellRadioGroup(Item, 0) = 1234 ' The 1234 is arbirary and it represents the
identifier for the radio group
End Sub

Private Sub Grid1_CellStateChanged(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As
Long)
   Debug.Print "In the 1234 radio group the """ & Grid1.Items.CellValue(,
Grid1.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_grid.GetItems();
m_grid.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_grid.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 OnCellStateChangedGrid1(long Item, long ColIndex)
{
    CItems items = m_grid.GetItems();
    long hCell = items.GetCellChecked( 1234 );
    COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
    OutputDebugString( V2S( &items.GetCellValue( 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 AxGrid1
    .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 AxGrid1_CellStateChanged(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_CellStateChangedEvent) Handles AxGrid1.CellStateChanged
    With AxGrid1.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:

```
axGrid1.BeginUpdate();
EXGRIDLib.Items items = axGrid1.Items;
for (int i = 0; i < items.ItemCount; i++)
{
    items.set_CellHasRadioButton(items[i], 0, true);
    items.set_CellRadioGroup(items[i], 0, 1234);
}
axGrid1.EndUpdate();
```

```
private void axGrid1_CellStateChanged(object sender,
AxEXGRIDLib._IGridEvents_CellStateChangedEvent e)
{
    string strOutput = axGrid1.Items.get_CellCaption( 0,
axGrid1.Items.get_CellChecked(1234) ).ToString();
    strOutput += " state = " + axGrid1.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.Grid1.BeginUpdate()
with thisform.Grid1.Items
    local i
```

```
    for i = 0 to .ItemCount - 1
        .DefaultItem = .ItemByIndex(i)
        .CellHasRadioButton( 0,0 ) = .t.
        .CellRadioGroup(0,0) = 1234
    next
endwith
thisform.Grid1.EndUpdate()
```

Note : The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
    .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
    .Items.CellBold(.Items(0), 0) = True
    .Items.CellBold(.Items(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True
    .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

Specifies the cell's extra data.

Type	Description
Item as Variant	A long expression that indicates the item's handle
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Variant	A variant expression that indicates the cell's user data.

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

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

Creates an gets the cell's built-in editor.

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 or key.
Editor	An Editor object being created or accessed

The CellEditor property gets you the custom cell editor if it exists, **else it creates it**. The CellEditor property **creates an empty editor if it wasn't created before**, so please pay attention when creating custom cell editors on the fly. You can use the HasCellEditor property to check whether a cell has associated a custom editor (created using the CellEditor property). You can have different type of editors in the same column using the CellEditor property. The CellEditor property builds a new editor for a specific cell. By default, the cell's editor is the default column's editor. Use the EditType property to specify an editor for the column. Use the DeleteCellEditor method to clear a particular cell editor created using the CellEditor property. Use the CellEditorVisible property to hide the cell's editor. You can use the BeginUpdate, EndUpdate method to refresh the control.

The following VB sample assigns a date type editor to the first cell:

```
With Grid1.Items
   Dim h As EXGRIDLibCtl.HITEM
   h = .ItemByIndex(0)
   If Not .HasCellEditor(h, 0) Then
      With .CellEditor(h, 0)
         .EditType = DateType
      End With
   End If
End With
```

The following VB sample assigns a spin type editor with min and max values:

```
With Grid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = SliderType
```

```
        .Option(exSliderWidth) = 0
        .Option(exSliderMin) = 5
        .Option(exSliderMax) = 10
    End With
End With
```

The following C++ sample assigns a date type editor to the first cell:

```
#include "Items.h"
#include "Editor.h"
void OnButton1()
{
    CItems items = m_grid.GetItems();
    COleVariant vtItem( items.GetItemByIndex( 0 ) ), vtColumn( (long)0 );
    if ( !items.GetHasCellEditor( vtItem, vtColumn ) )
    {
        CEditor editor = items.GetCellEditor( vtItem, vtColumn );
        editor.SetEditType( 7 /*DateType*/ );
    }
}
```

The following C++ sample assigns a spin type editor with min and max values:

```
#include "Items.h"
#include "Editor.h"
CItems items = m_grid.GetItems();
CEditor editor = items.GetCellEditor( COleVariant( items.GetFirstVisibleItem() ),
COleVariant( long(0) ) );
editor.SetEditType( 20 /*SliderType*/ );
editor.SetOption( 41 /*exSliderWidth */, COleVariant( long(0) ) );
editor.SetOption( 43 /*exSliderMin*/, COleVariant( long(5) ) );
editor.SetOption( 44 /*exSliderMax*/, COleVariant( long(10) ) );
```

The following VB.NET sample assigns a date type editor to the first cell:

```
With AxGrid1.Items
    Dim h As Integer = .ItemByIndex(0)
    If Not .HasCellEditor(h, 0) Then
        With .CellEditor(h, 0)
```

```
        .EditType = EXGRIDLib.EditTypeEnum.DateType
      End With
    End If
End With
```

The following VB.NET sample assigns a spin type editor with min and max values:

```
With AxGrid1.Items
   With .CellEditor(.FirstVisibleItem, 0)
      .EditType = EXGRIDLib.EditTypeEnum.SliderType
      .Option(EXGRIDLib.EditorOptionEnum.exSliderWidth) = 0
      .Option(EXGRIDLib.EditorOptionEnum.exSliderMin) = 5
      .Option(EXGRIDLib.EditorOptionEnum.exSliderMax) = 10
   End With
End With
```

The following C# sample assigns a date type editor to the first cell:

```
int h = axGrid1.Items[0];
if ( !axGrid1.Items.get_HasCellEditor( h, 0 ) )
{
   EXGRIDLib.Editor editor = axGrid1.Items.get_CellEditor( h, 0 );
   if ( editor != null )
      editor.EditType = EXGRIDLib.EditTypeEnum.DateType;
}
```

The following C# sample assigns a spin type editor with min and max values:

```
EXGRIDLib.Editor editor = axGrid1.Items.get_CellEditor(axGrid1.Items.FirstVisibleItem, 0);
editor.EditType = EXGRIDLib.EditTypeEnum.SliderType;
editor.set_Option(EXGRIDLib.EditorOptionEnum.exSliderWidth, 0);
editor.set_Option(EXGRIDLib.EditorOptionEnum.exSliderMin, 5);
editor.set_Option(EXGRIDLib.EditorOptionEnum.exSliderMax, 10);
```

The following VFP sample assigns a date type editor to the first cell:

```
with thisform.Grid1.Items
   thisform.Grid1.BeginUpdate()
   .DefaultItem = .ItemByIndex( 0 )
```

```
    if ( !.HasCellEditor( 0, 0 ) )
        .CellEditor( 0, 0 ).EditType = 7
    endif
    thisform.Grid1.EndUpdate()
endwith
```

The following VFP sample assigns a spin type editor with min and max values:

```
with thisform.Grid1.Items
    with .CellEditor(.FirstVisibleItem, 0)
        .EditType = 20 && SliderType
        .Option(41) = 0  && exSliderWidth
        .Option(43) = 5 && exSliderMin
        .Option(44) = 10 && exSliderMax
    endwith
endwith
```

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

Specifies whether a column's editor is visible or hidden into the cell.

Type	Description
Item as Variant	A long expression that indicates the item's handle
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key
EditorVisibleEnum	An EditorVisibleEnum expression that indicates whether the cell's editor visible, always visible or hidden.

Use the CellEditorVisible property to hide the cell's editor. Use the Editor or CellEditor property to assign an editor to the entire column or to a specific cell. Use the Locked property to lock an editor. If the cell's editor is hidden, the cell displays the CellValue property as a plain text, if the CellValueFormat property is exText, else if the CellValueFormat property is exHTML the cell displays the CellValue using built-in HTML format.

The following VB sample hides the editor for the focused cell:

```
With Grid1.Items
   .CellEditorVisible(.FocusItem, Grid1.FocusColumnIndex) = False
End With
```

The following C++ sample hides the editor for the focused cell:

```
#include "Items.h"
CItems items = m_grid.GetItems();
items.SetCellEditorVisible( COleVariant( items.GetFocusItem() ), COleVariant(
m_grid.GetFocusColumnIndex() ), FALSE );
```

The following VB.NET sample hides the editor for the focused cell:

```
With AxGrid1.Items
   .CellEditorVisible(.FocusItem, AxGrid1.FocusColumnIndex) = False
End With
```

The following C# sample hides the editor for the focused cell:

```
EXGRIDLib.Items items = axGrid1.Items;
items.set_CellEditorVisible(items.FocusItem, axGrid1.FocusColumnIndex, false);
```

The following VFP sample hides the editor for the focused cell:

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .CellEditorVisible( 0, thisform.Grid1.FocusColumnIndex ) = .f.
endwith
```

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

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

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression that indicates whether the cell is enabled or disabled.

Use the CellEnabled property to disable a cell. A disabled cell looks grayed. Use the EnableItem property to disable an item. Once that one cell is disabled it cannot be checked or clicked. Use the SelectableItem property to specify the user can select an item. To disable a column you can use Enabled property of the Column object.

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

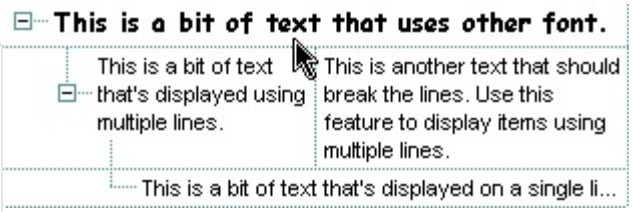
property Items.CellFont ([Item as Variant], [ColIndex as Variant]) as IFontDisp

Retrieves or sets the cell's font.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or the column's key.
IFontDisp	A Font object that indicates the item's font.

By default, the CellFont property is nothing. If the CellFont property is 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 CellValueFormat 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 ItemHeight property to specify the height of the item.

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




```
With Grid1.Items
   .CellFont(.FocusItem, Grid1.FocusColumnIndex) =
Grid1.Font
   With .CellFont(.FocusItem, 0)
      .Name = "Comic Sans MS"
      .Size = 10
      .Bold = True
   End With
End With
Grid1.Refresh
```

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

```
#include "Items.h"
#include "Font.h"
CItems items = m_grid.GetItems();
COleVariant vtItem(items.GetFocusItem()), vtColumn(
(long)m_grid.GetFocusColumnIndex() );
```

```
items.SetCellFont( vtItem, vtColumn, m_grid.GetFont().m_lpDispatch );
COleFont font = items.GetCellFont( vtItem, vtColumn );
font.SetName( "Comic Sans MS" );
font.SetBold( TRUE );
m_grid.Refresh();
```

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

```
With AxGrid1.Items
   .CellFont(.FocusItem, AxGrid1.FocusColumnIndex) = IFDH.GetIFontDisp(AxGrid1.Font)
   With .CellFont(.FocusItem, 0)
      .Name = "Comic Sans MS"
      .Bold = True
   End With
End With
AxGrid1.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:

```
axGrid1.Items.set_CellFont(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex ,
IFDH.GetIFontDisp(axGrid1.Font));
stdole.IFontDisp spFont = axGrid1.Items.get_CellFont(axGrid1.Items.FocusItem,
axGrid1.FocusColumnIndex);
spFont.Name = "Comic Sans MS";
```

```
spFont.Bold = true;
axGrid1.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.Grid1.Items
    .DefaultItem = .FocusItem
    .CellFont(0,thisform.Grid1.FocusColumnIndex) = thisform.Grid1.Font
    with .CellFont(0,0)
        .Name = "Comic Sans MS"
        .Bold = .t.
    endwith
endwith
thisform.Grid1.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 cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Color	A color expression that indicates the cell's foreground color

The CellForeColor property identifies the cell's foreground color. Use the ClearCellForeColor property to clear the cell's foreground color. Use the ItemForeColor property to specify the the item's foreground color. Use the Def(exCellForeColor) property to specify the foreground color for all cells in the column. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

For instance, the following VB code changes the left top cell of your control:
Grid1.Items.CellForeColor(Grid1.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 VB sample changes the foreground color for the focused cell:

```
With Grid1.Items
   .CellForeColor(.FocusItem, Grid1.FocusColumnIndex) = vbRed
End With
```

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

```
axGrid1.Items.set_CellForeColor(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex,
ToUInt32(Color.Red));
```

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

```
With AxGrid1.Items
   .CellForeColor(.FocusItem, AxGrid1.FocusColumnIndex) = ToUInt32(Color.Red)
End With
```

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

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

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

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .CellForeColor( 0, thisform.Grid1.FocusColumnIndex ) = RGB(255,0,0)
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

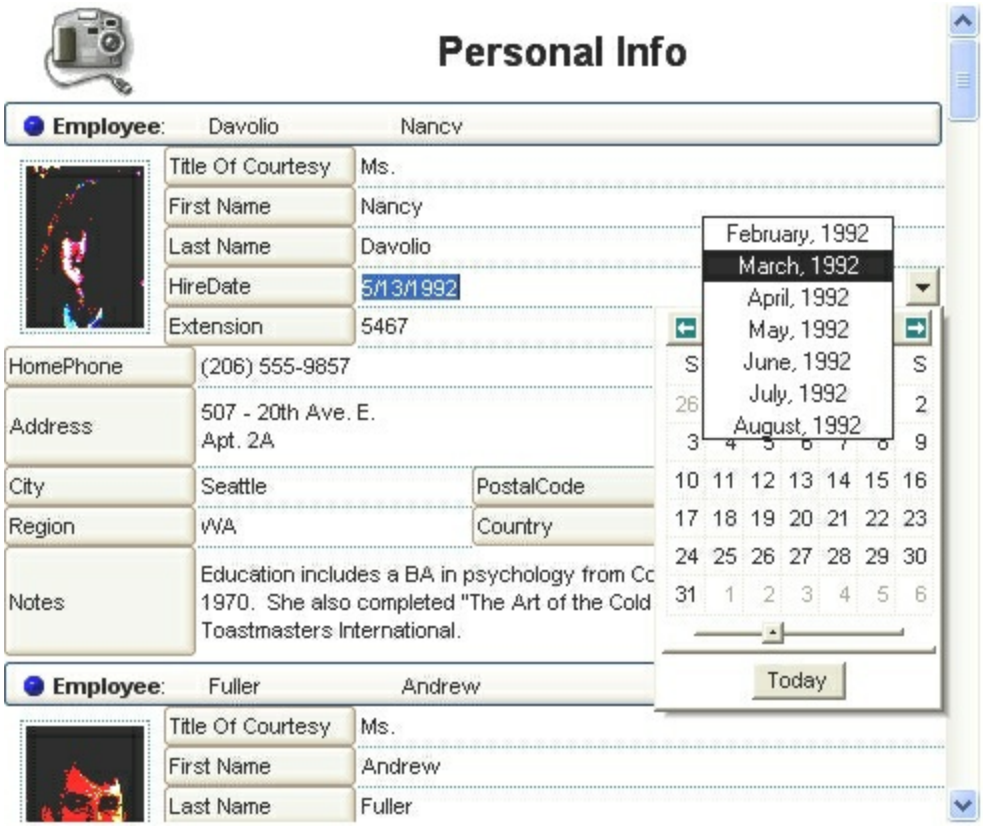
```
With Grid1
    .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
    .Items.CellBold(.Items(0), 0) = True
    .Items.CellBold(.Items(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True
    .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

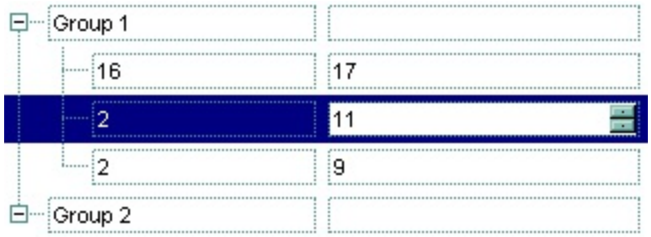
Specifies the arrangement of the fields inside the cell.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
String	A CRD string expression that indicates the layout of the cell. The Index elements in the CRD string indicates the index of the column being displayed.

By default, the CellFormatLevel property is empty. If the CellFormatLevel property is empty, the cell displays it's caption. Use the CellValue property to assign a value to a cell. If the CellFormatLevel property is not empty, it indicates the layout being displayed in the cell's area. For instance, the CellFormatLevel = "1/2" indicates that the cell's area is vertically divided such as the up part displays the caption of the cell in the first column, and the down part displays the caption of the cell in the second column. The height of the item is NOT changed, after calling the CellFormatLevel property. Use the ItemHeight property to specify the height of the item. Use the DefaultItemHeight property to specify the default height of the items before inserting them. Use the Def(exCellFormatLevel) property to specify the layout for all cells in the same column. For instance, you can have a specify layout for some cells using the Def(exCellFormatLevel) property (by default it is applied to all cells in the column), and for other cells you can use the CellFormatLevel property to specify different layouts, or to remove the default layout. Use the FormatLevel property to arrange the columns in the control's header bar.



For instance, this layout [dgl=1]""[b=0]:4,(4;""[b=4]/0/4;""[b=1]),""[b=0]:4 adds a 4 pixels-borders around any object its applies (in this case all columns), like in the following picture:



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. Use the Def(exCellDrawPartsOrder) property to specify the order of the drawing parts inside the cell.

The following VB sample right aligns the focused cell:

```
With Grid1.Items
   .CellHAlignment(.FocusItem, Grid1.FocusColumnIndex) =
AlignmentEnum.RightAlignment
End With
```

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

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

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

```
With AxGrid1.Items
   .CellHAlignment(.FocusItem, AxGrid1.FocusColumnIndex) =
```

```
EXGRIDLib.AlignmentEnum.RightAlignment
End With
```

The following C# sample right aligns the focused cell:

```
axGrid1.Items.set_CellHAlignment(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex,
EXGRIDLib.AlignmentEnum.RightAlignment);
```

The following VFP sample right aligns the focused cell:

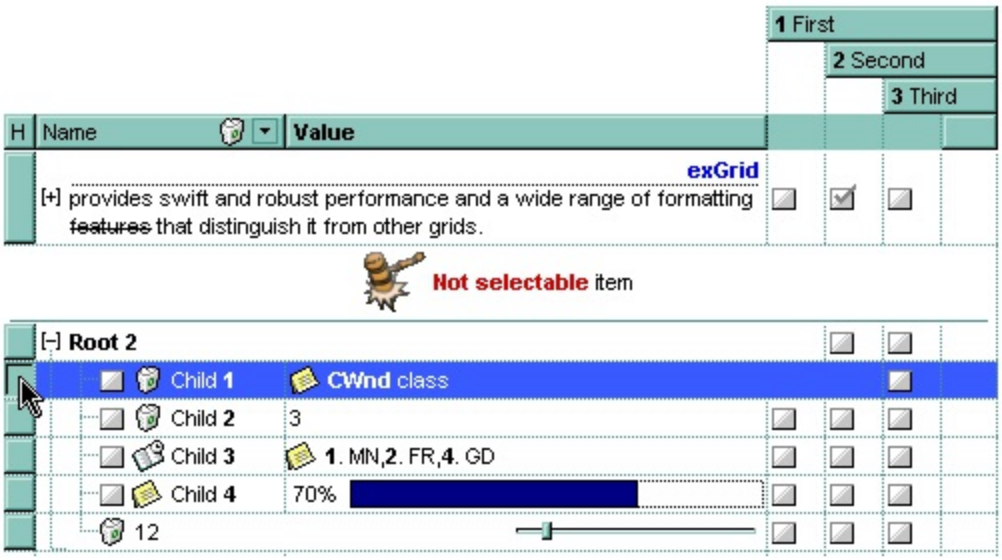
```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .CellHAlignment(0,thisform.Grid1.FocusColumnIndex) = 2 && RightAlignment
endwith
```

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

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

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression that indicates whether the cell contains a button.

Each time when user clicks a cell that has CellHasButton property to True, the control fires ButtonClick event. The caption of the push button is defined by the CellValue property. Use the Def property to assign buttons to all cells in the column. Use the CellButtonAutoWidth property to specify whether the buttons fit the cell's content. Use the CellEditor property to assign an editor to a cell. Use the Editor property to assign the same editor to all cells in the column. Use the AddButton method to add a new button to an editor. If you need multiple buttons inside the same cell, you can split the cell in multiple pieces and add a button to each piece. Use the SplitCell property to split a cell. Use the Background(exCellButtonUp) or Background(exCellButtonDown) property to change the visual appearance for the buttons in the control.



The following VB sample adds a header row column:

```
With Grid1.Columns.Add("H")
  .Def(exCellHasButton) = True
  .Position = 0
```

```
    .AllowDragging = False
    .HeaderAlignment = CenterAlignment
    .Width = 16
End With
```

The following VB sample assigns a button for each cell in the first column, as soon as a new item is inserted:

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
    Grid1.Items.CellHasButton(Item, 0) = True
End Sub

Private Sub Grid1_ButtonClick(ByVal Item As EXGRIDLibCtl.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 Grid1.Items
    .CellHasButton(.FocusItem, Grid1.FocusColumnIndex) = True
End With
```

The following VB sample adds four buttons in the same cell:

```
With Grid1.Items
    Dim h As HITEM
    h = .FirstVisibleItem()
    .CellValue(h, 0) = "B1"
    .CellHasButton(h, 0) = True
    f = .SplitCell(h, 0)
    .CellValue(, f) = "B2"
    .CellHasButton(, f) = True
    f = .SplitCell(, f)
    .CellValue(, f) = "B3"
    .CellHasButton(, f) = True
    f = .SplitCell(, f)
    .CellValue(, f) = "B4"
    .CellHasButton(, f) = True
End With
```

The following C++ sample adds a header row column:

```
#include "Column.h"
#include "Columns.h"
CColumns columns = m_grid.GetColumns();
CColumn column( V_DISPATCH( &columns.Add("H") ) );
column.SetHeaderAlignment( 1 );
column.SetDef(2, COleVariant( VARIANT_TRUE ) );
column.SetPosition( 0 );
column.SetWidth( 16 );
column.SetAllowDragging( FALSE );
```

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

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

The following VB.NET sample adds a header row column:

```
With AxGrid1.Columns.Add("H")
   .Def(EXGRIDLib.DefColumnEnum.exCellHasButton) = True
   .Position = 0
   .AllowDragging = False
   .HeaderAlignment = EXGRIDLib.AlignmentEnum.CenterAlignment
   .Width = 16
End With
```

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

```
With AxGrid1.Items
   .CellHasButton(.FocusItem, AxGrid1.FocusColumnIndex) = True
End With
```

The following C# sample adds a header row column:

```
EXGRIDLib.Columns columns = axGrid1.Columns;
EXGRIDLib.Column column = columns.Add("H") as EXGRIDLib.Column;
```

```
column.set_Def(EXGRIDLib.DefColumnEnum.exCellHasButton, true);
column.Position = 0;
column.HeaderAlignment = EXGRIDLib.AlignmentEnum.CenterAlignment;
column.AllowDragging = false;
column.Width = 16;
```

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

```
axGrid1.Items.set_CellHasButton(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex,
true);
```

The following VFP sample adds a header row column:

```
with thisform.Grid1.Columns.Add("H")
   .Position = 0
   .Def(2) = .t.
   .AllowDragging = .f.
   .Width = 16
endwith
```

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

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .CellHasButton(0,thisform.Grid1.FocusColumnIndex) = .t.
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
```

```
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True
    .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

Retrieves or sets a value indicating whether the cell has an associated checkbox.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression that indicates whether the cell contains a check box button.

Use the CellState property to change the state of the cell of the check box type. 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. Use the CellHasRadioButton property to add a radio button to your cell. Use the PartialCheck property to enable partial check feature. The CheckImage property specifies the icon being displayed for different check box states. The EditType.CheckValueType value specifies whether the editor displays a check box to represent the cell's value. Use the Def property to assign check boxes for all cells in the column. Use the CellImage property to add a single icon to a cell. Use the CellImages property to assign multiple icons to a cell. Use the CellPicture property to load a custom size picture to a cell. Use the FilterType property on exCheck to filter for checked or unchecked items. Use the Def(exCellDrawPartsOrder) property to specify the order of the drawing parts inside the cell.

The following VB sample assigns a checkbox to all cells in the first column:

```
With Grid1.Columns(0)
  .Def(exCellHasCheckBox) = True
End With
```

The following VB sample assigns a checkbox to all cells in the first column:

```
Dim h As Variant
Grid1.BeginUpdate
With Grid1.Items
For Each h In Grid1.Items
  .CellHasCheckBox(h) = True
Next
End With
```

```
Grid1.EndUpdate
```

The following VB sample uses the AddItem event to add a check box for all cells in the first column:

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
   Grid1.Items.CellHasCheckBox(Item) = True
End Sub
```

The VB following sample uses the CellStateChanged event to display a message when a cell of radio or check type has changed its state:

```
Private Sub Grid1_CellStateChanged(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As Long)
   Debug.Print "The cell """ & Grid1.Items.CellValue(Item, ColIndex) & """ has changed its state. The new state is " & IIf(Grid1.Items.CellState(Item, ColIndes) = 0, "Unchecked", "Checked")
End Sub
```

The EditType.CheckValueType value specifies whether the editor displays a check box to represent the cell's value. For instance, if you have a column of boolean values, you can use the following VB code to assign check boxes to each cell in the column:

```
With Grid1.Columns("Check").Editor
      .EditType = CheckValueType
End With
```

In this case, the CellValue property for each cell in the column specifies the state of the check box as follows:

- 0 (unchecked state). The control displays the unchecked state check box.
- 1 (checked state). The control displays the checked state check box.
- 2 (partial checked state). The control displays the partial checked state check box.

If the values in the column differs than 0, 1, or 2 you can call the Editor.Option/DefaultEditorOption property to specify the check box states being displayed. For instance, if your column contains boolean values, True (-1) and False (0), you can use the following sample to get displayed the checked states instead partial checked states.

```
With Grid1.Columns("Check").Editor
      .EditType = CheckValueType
```

```
    .Option(exCheckValue2) = 1
End With
```

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

```
With Grid1.Items
   .CellHasCheckBox(.FocusItem, Grid1.FocusColumnIndex) = True
End With
```

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

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

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

```
axGrid1.Items.set_CellHasCheckBox(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex,
true);
```

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

```
With AxGrid1.Items
   .CellHasCheckBox(.FocusItem, AxGrid1.FocusColumnIndex) = True
End With
```

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

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .CellHasCheckBox(0,thisform.Grid1.FocusColumnIndex) = .t.
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
    .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
    .Items.CellBold(.Items(0), 0) = True
    .Items.CellBold(.Items(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True
    .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

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

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression that indicates whether the cell contains a radio button.

Use the CellState property to change the state of the cell of the radio type. 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. Call the CellHasCheckBox property to add a check box to the cell. Use the CellRadioGroup property To group or ungroup cells of radio type. Use the Def property to assign radio buttons to 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. Use the Def(exCellDrawPartsOrder) property to specify the order of the drawing parts inside the cell.

The following VB sample assigns a radio button to all cells in the first column:

```
With Grid1.Columns(0)
   .Def(exCellHasRadioButton) = True
End With
```

The following VB sample assigns a radio button to all cells in the first column, and groups all of them in the same radio group (1234):

```
Dim h As Variant
Grid1.BeginUpdate
With Grid1.Items
For Each h In Grid1.Items
   .CellHasRadioButton(h, 0) = True
   .CellRadioGroup(h, 0) = 1234
Next
End With
```

```
Grid1.EndUpdate
```

The following VB sample assigns a radio button to all cells in the first column, when adding a new item:

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
   Grid1.Items.CellHasRadioButton(Item, 0) = True
   Grid1.Items.CellRadioGroup(Item, 0) = 1234
End Sub
```

Call the Grid1.Items.CellValue(, Grid1.Items.CellChecked(1234)) to get the cell's value that's checked in the group 1234.

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

```
With Grid1.Items
   .CellHasRadioButton(.FocusItem, Grid1.FocusColumnIndex) = True
   .CellRadioGroup(.FocusItem, Grid1.FocusColumnIndex) = 1234
End With
```

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

```
#include "Items.h"
CItems items = m_grid.GetItems();
COleVariant vtItem(items.GetFocusItem()), vtColumn(m_grid.GetFocusColumnIndex());
items.SetCellHasRadioButton( vtItem, vtColumn, TRUE );
items.SetCellRadioGroup( vtItem, vtColumn, 1234 );
```

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

```
With AxGrid1.Items
   .CellHasRadioButton(.FocusItem, AxGrid1.FocusColumnIndex) = True
   .CellRadioGroup(.FocusItem, AxGrid1.FocusColumnIndex) = 1234
End With
```

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

```
axGrid1.Items.set_CellHasRadioButton(axGrid1.Items.FocusItem,
axGrid1.FocusColumnIndex, true);
axGrid1.Items.set_CellRadioGroup(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex,
```

```
1234);
```

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

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .CellHasRadioButton(0,thisform.Grid1.FocusColumnIndex) = .t.
   .CellRadioGroup(0,thisform.Grid1.FocusColumnIndex) = 1234
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

Specifies whether the cell 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 cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression that indicates whether the cell is of hyper link type.

A cell that has CellHyperLink property to True, is a cell of hyper link type. Use the CellHyperLink property to add hyper links to your control. If the user clicks a cell of hyper link type, the control fires the HyperLinkClick event. Use the CellForeColor property to change the cell's foreground color. Use the HyperLinkColor property to change the color that's used by control when the cursor is over a cell of hyper link type.

If you would like a HOT TRACKING sample you could use the ItemFromPoint property like in the following sample. The sample changes the foreground and the background color for the tracking item. Use the ItemBackColor property to change the item's background color. Use the ClearItemBackColor property to clear the item's background color, after setting using the ItemBackColor property. Use the ItemForeColor property to change the item's foreground color. Use the ClearItemForeColor property to clear the item's foreground color.

```
Private Declare Function SetCapture Lib "user32" (ByVal hwnd As Long) As Long
Private Declare Function ReleaseCapture Lib "user32" () As Long
Dim hIA As HITEM

Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   Dim hI As HITEM, c As Long, hit As HitTestInfoEnum
   With Grid1
   hI = .ItemFromPoint(-1, -1, c, hit)
   If (hI = 0) Then
      ReleaseCapture
   End If
   If Not hI = hIA Then
      With .Items
```

```
        If Not hIA = 0 Then
            .ClearItemBackColor (hIA)
            .ClearItemForeColor (hIA)
        End If
        If Not hI = 0 Then
            SetCapture Grid1.hwnd
            .ItemBackColor(hI) = vbRed
            .ItemForeColor(hI) = vbWhite
        End If
    End With
    hIA = hI
  End If
  End With
End Sub
```

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

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

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Long	A long value that indicates the index of the icon in Images collection. The Images collection is 1 based. The last 7 bits in the high significant byte of the long expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part.

The CellImage property assigns a single icon to a cell. Use the CellImage() = 0 to remove the cell's icon, that was previous assigned using the CellImage property . Use the CellImages property to assign multiple icons to a single cell. The CellImageClick event occurs when the user clicks the cell's icon. The icon's size is always 16 x 16. Use the CellPicture property to load a a picture of different size. Use the Images or ReplaceIcon method to load icons to the control. 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 FilterType property on exImage to filter items by icons. Use the <img> HTML tag to insert icons inside the cell's caption. Use the Def(exCellDrawPartsOrder) property to specify the order of the drawing parts inside the cell. The ImageSize property defines the size (width/height) of the icons within the control's Images collection.

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

```
With Grid1.Items
   .CellImage(.FocusItem, Grid1.FocusColumnIndex) = 1
End With
```

The following VB sample changes the cell's icon for all icons in the first column, as soon as new items are inserted to the control:

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
```

```
   Grid1.Items.CellImage(Item, 0) = 1
End Sub
```

The VB following sample changes the cell's image when the user clicks on the cell's image (to run the following sample you have to add two images to the grid's images collection),

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
   Grid1.Items.CellImage(Item, 0) = 1
End Sub

Private Sub Grid1_CellImageClick(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As
Long)
   Grid1.Items.CellImage(Item, ColIndex) = Grid1.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_grid.GetItems();
items.SetCellImage( COleVariant( items.GetFocusItem() ), COleVariant(
(long)m_grid.GetFocusColumnIndex() ), 1 );
```

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

```
axGrid1.Items.set_CellImage(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex, 1);
```

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

```
With AxGrid1.Items
   .CellImage(.FocusItem, AxGrid1.FocusColumnIndex) = 1
End With
```

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

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .CellImage(0,thisform.Grid1.FocusColumnIndex) = 1
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

Specifies an additional list of icons shown in the cell.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the column's index, a string expression that indicates the column's caption or the column's key.
Variant	A string expression that indicates the list of icons shown in the cell. For instance, the "1,2,3" indicates that the icons 1, 2, 3 are displayed in the 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 Images or ReplaceIcon method to assign icons at runtime. Use the Def(exCellDrawPartsOrder) property to specify the order of the drawing parts inside the cell. The ImageSize property defines the size (width/height) of the icons within the control's Images collection.

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

```
With Grid1.Items
    .CellImages(.ItemByIndex(0), 1) = "1,3"
End With
```

The following VB sample displays the index of icon being clicked, when the cell contains multiple icons (MouseUp event):

```
Private Sub Grid1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
  Dim i As HITEM, h As HitTestInfoEnum, c As Long
  With Grid1
    i = .ItemFromPoint(-1, -1, c, h)
  End With
  If (i <> 0) Then
    If exHTCellIcon = (h And exHTCellIcon) Then
      Debug.Print "The index of icon being clicked is: " & CLng((h And &HFFFF0000) /
65536)
    End If
```

```
    End If
End Sub
```

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

```
#include "Items.h"
CItems items = m_grid.GetItems();
items.SetCellImages( COleVariant( items.GetFocusItem() ), COleVariant(
(long)m_grid.GetFocusColumnIndex() ), COleVariant( "1,3" ) );
```

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

```
#include "Items.h"
void OnMouseUpGrid1(short Button, short Shift, long X, long Y)
{
    CItems items = m_grid.GetItems();
    long c = 0, hit = 0, h = m_grid.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 AxGrid1.Items
    .CellImages(.FocusItem, AxGrid1.FocusColumnIndex) = "1,3"
End With
```

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

```
Private Sub AxGrid1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_MouseUpEvent) Handles AxGrid1.MouseUpEvent
    With AxGrid1
```

```
    Dim i As Integer, c As Integer, hit As EXGRIDLib.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:

```
axGrid1.Items.set_CellImages(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex, "1,3");
```

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

```
private void axGrid1_MouseUpEvent(object sender,
AxEXGRIDLib._IGridEvents_MouseUpEvent e)
{
  int c = 0;
  EXGRIDLib.HitTestInfoEnum hit;
  int i = axGrid1.get_ItemFromPoint(e.x, e.y, out c, out hit);
  if ((i != 0))
  {
    if ((Convert.ToUInt32(hit) &
Convert.ToUInt32(EXGRIDLib.HitTestInfoEnum.exHTCellIcon)) ==
Convert.ToUInt32(EXGRIDLib.HitTestInfoEnum.exHTCellIcon))
    {
      string s = axGrid1.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.Grid1.Items
  .DefaultItem = .FocusItem
  .CellImages(0,thisform.Grid1.FocusColumnIndex) = "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.Grid1
   .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 specifies whether the cell should appear in italic.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression that indicates whether the cell should appear in italic.

Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the CellValueFormat 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 Grid1.Items
   .CellItalic(.FocusItem, Grid1.FocusColumnIndex) = True
End With
```

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

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

The following C# sample makes italic the focused cell:

```
axGrid1.Items.set_CellItalic(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex, true);
```

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

```
With AxGrid1.Items
   .CellItalic(.FocusItem, AxGrid1.FocusColumnIndex) = True
```

```
End With
```

The following VFP sample makes italic the focused cell:

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .CellItalic( 0, thisform.Grid1.FocusColumnIndex() ) = .t.
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

property Items.CellItem (Cell as HCELL) as HITEM

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

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

Use the CellItem property to retrieve the item's handle. Use the ItemCell property to 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.

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

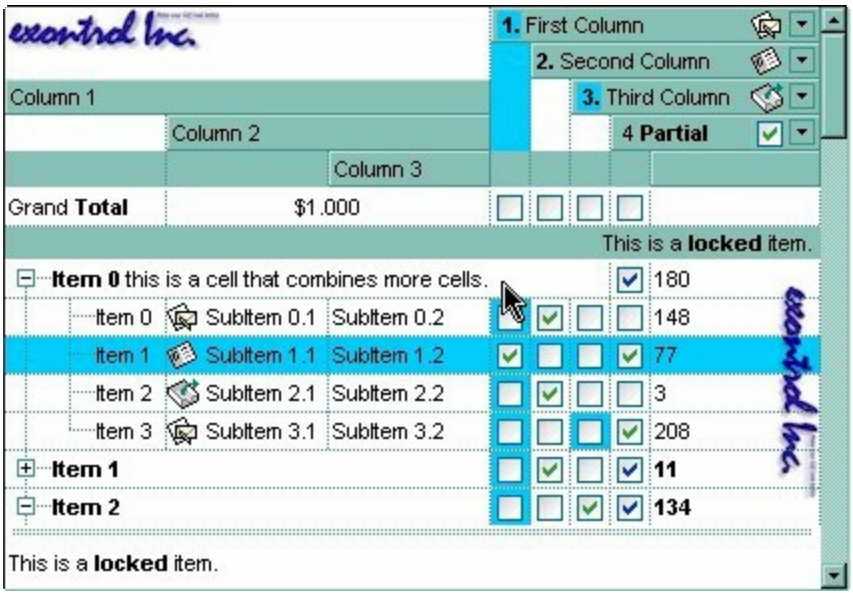
```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

Retrieves or sets a value that indicates the index of the cell that's merged to.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the column's index, a string expression that indicates the column's caption or the column's key.
Variant	A long expression that indicates the index of the cell that's merged with, a safe array that holds the indexes of the cells being merged.

Use the CellMerge property to combine two or more cells in the same item in a single cell. The data of the source cell is displayed in the new larger cell. All the other cells' data is not lost. Use the ItemDivider property to display a single cell in the entire item. Use the UnmergeCells method to unmerge the merged cells. Use the CellMerge property to unmerge a single cell. Use the MergeCells method to combine one or more cells in a single cell. Use the SplitCell property to split a cell. Use the Add method to add new columns to the control.



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

```
With Grid1.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_grid.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 AxGrid1.Items
   .CellMerge(.FocusItem, 0) = 1
   .CellMerge(.FocusItem, 0) = 2
End With
```

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

```
axGrid1.Items.set_CellMerge(axGrid1.Items.FocusItem, 0, 1);
axGrid1.Items.set_CellMerge(axGrid1.Items.FocusItem, 0, 2);
```

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

```
with thisform.Grid1.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

The following sample shows few methods to unmerge cells:

```
With Grid1
   With .Items
      .UnmergeCells .ItemCell(.RootItem(0), 0)
   End With
End With
```

```
With Grid1
```

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

```
With Grid1
    .BeginUpdate
    With .Items
        .CellMerge(.RootItem(0), 0) = -1
        .CellMerge(.RootItem(0), 1) = -1
        .CellMerge(.RootItem(0), 2) = -1
    End With
    .EndUpdate
End With
```

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

```
 With Grid1
    With .Items
        .CellMerge(.RootItem(0), 0) = Array(1, 2)
    End With
End With
```

```
With Grid1
    .BeginUpdate
    With .Items
        Dim r As Long
        r = .RootItem(0)
        .CellMerge(r, 0) = 1
        .CellMerge(r, 0) = 2
    End With
    .EndUpdate
End With
```

```
With Grid1
    .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 Grid1
    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 Grid1
    With .Items
        Dim r As Long
        r = .RootItem(0)
        .MergeCells Array(.ItemCell(r, 0), .ItemCell(r, 1), .ItemCell(r, 2))
    End With
End With
```

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

Specifies the cell's owner draw handler.

Type	Description
Item as Variant	A long expression that indicates the item's handle being painted
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
IOwnerDrawHandler	An object that implements the IOwnerDrawHandler interface

Use the CellOwnerDraw property to paint yourself the cell. The CellOwnerDraw property specifies whether the user is responsible for painting the cell. By default, the CellOwnerDraw property is nothing, and so the control does the painting. Using the notification interfaces is faster than using events. For instance, let's say that we need cells where for some values we need to get displayed other things. Use the Def(exCellOwneDraw) property to assign an owner draw object for the entire column.

The following VB sample displays the CellValue when it is different than 5, and displays another string when the CellValue property is 5:

```
Implements IOwnerDrawHandler
Private Type RECT
   left As Long
   top As Long
   right As Long
   bottom As Long
End Type
Private Declare Function DrawText Lib "user32" Alias "DrawTextA" (ByVal hdc As Long,
ByVal lpStr As String, ByVal nCount As Long, lpRect As RECT, ByVal wFormat As Long) As
Long
Private Const DT_VCENTER = &H4
Private Const DT_SINGLELINE = &H20

Private Sub Form_Load()
   With Grid1.Items
     .CellValue(.FirstVisibleItem, 0) = 5
```

```
    Set .CellOwnerDraw(.FirstVisibleItem, 0) = Me
  End With
End Sub

Private Sub IOwnerDrawHandler_DrawCell(ByVal hdc As Long, ByVal left As Long, ByVal
top As Long, ByVal right As Long, ByVal bottom As Long, ByVal Item As Long, ByVal
Column As Long, ByVal Source As Object)
  Dim s As String
  s = Source.Items.CellValue(Item, Column)
  If (s = "5") Then
    s = "just another string"
  End If
  Dim r As RECT
  r.left = left + 2
  r.right = right - 2
  r.top = top
  r.bottom = bottom
  DrawText hdc, s, Len(s), r, DT_VCENTER Or DT_SINGLELINE
End Sub

Private Sub IOwnerDrawHandler_DrawCellBk(ByVal hdc As Long, Options As Variant, ByVal
left As Long, ByVal top As Long, ByVal right As Long, ByVal bottom As Long, ByVal Item As
Long, ByVal Column As Long, ByVal Source As Object)

End Sub
```

The following VB6 sample displays half of the cell's background in green:

```
Implements IOwnerDrawHandler
Private Type RECT
  left As Long
  top As Long
  right As Long
  bottom As Long
End Type
Private Declare Function DrawText Lib "user32" Alias "DrawTextA" (ByVal hdc As Long,
ByVal lpStr As String, ByVal nCount As Long, lpRect As RECT, ByVal wFormat As Long) As
```

```
Long
Private Const DT_VCENTER = &H4
Private Const DT_SINGLELINE = &H20

Private Declare Function FillRect Lib "user32" (ByVal hdc As Long, lpRect As RECT, ByVal
hBrush As Long) As Long
Private Declare Function CreateSolidBrush Lib "gdi32" (ByVal crColor As Long) As Long
Private Declare Function DeleteObject Lib "gdi32" (ByVal hObject As Long) As Long



Private Sub Form_Load()
   With Grid1.Items
     Set .CellOwnerDraw(.FirstVisibleItem, 0) = Me
   End With
End Sub

Private Sub IOwnerDrawHandler_DrawCell(ByVal hdc As Long, ByVal left As Long, ByVal
top As Long, ByVal right As Long, ByVal bottom As Long, ByVal Item As Long, ByVal
Column As Long, ByVal Source As Object)
   Dim s As String
   s = Source.Items.CellValue(Item, Column)
   Dim r As RECT
   r.left = left + 2
   r.right = right - 2
   r.top = top
   r.bottom = bottom
   DrawText hdc, s, Len(s), r, DT_VCENTER Or DT_SINGLELINE
End Sub

Private Sub IOwnerDrawHandler_DrawCellBk(ByVal hdc As Long, Options As Variant, ByVal
left As Long, ByVal top As Long, ByVal right As Long, ByVal bottom As Long, ByVal Item As
Long, ByVal Column As Long, ByVal Source As Object)
   Dim r As RECT
   r.left = left + 2
   r.right = right - 2
   r.top = top
   r.bottom = bottom
```

```
    r.right = (r.left + r.right) / 2
    Dim brush As Long
    brush = CreateSolidBrush(rgb(0, 255, 0))
        FillRect hdc, r, brush
    DeleteObject brush

End Sub
```

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. Use the ItemCell property to get a master cell giving the handle of the item and the index of the column. The CellParent property gets 0 if the cell is the master cell, 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 EXGRIDLibCtl.Grid, ByVal h As EXGRIDLibCtl.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 EXGRIDLibCtl.Grid, ByVal h As EXGRIDLibCtl.HITEM,
ByVal c As Long) As EXGRIDLibCtl.HCELL
   With g.Items
      Dim r As EXGRIDLibCtl.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( CGrid grid, long hItem, long nColIndex )
{
    return V2I( &grid.GetItems().GetCellParent( COleVariant( hItem ), COleVariant( nColIndex
) ) ) == 0;
}
```

The following C++ sample determines the master cell (the cell from where the splitting starts):

```
long getMaster( CGrid grid, long hItem, long nColIndex )
```

```
{
   COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
   CItems items = grid.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 AxEXGRIDLib.AxGrid, 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 AxEXGRIDLib.AxGrid, 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(AxEXGRIDLib.AxGrid grid, int h, int c)
{
    return Convert.ToInt32(grid.Items.get_CellParent(h, c)) != 0;
}
```

The following C# sample determines the master cell (the cell from where the splitting starts):

```
private long getMaster(AxEXGRIDLib.AxGrid 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 the cell's picture.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Variant	A Picture object 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 property specifies the width in pixels of the cell's picture. If it is not specified, the picture's size determines the width to paint the picture inside the cell. The CellPictureHeight property specifies the height in pixels of the cell's picture. If it is not specified, the picture's size determines the height to paint the picture inside the cell. Use the <img> built-in HTML tag to insert multiple custom size picture to the same cell. Use the Def(exCellDrawPartsOrder) property to specify the order of the drawing parts inside the cell.

The following VB samples loads picture from a file:

```
Grid1.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 +
```

"XgBadlDXdYSXRb9wWBclK2taF1gAI5HiPaN8oPdlNWbaF23KAwyWkNYyXxg9p3WNYjU/c

```
With Grid1
   .BeginUpdate
   .BackColor = vbWhite
   .Columns.Add "Column 1"
   .Items.CellPicture(.Items.AddItem("Item 1"), 0) = s
   .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_grid.GetItems();
  items.SetCellPicture( COleVariant( items.GetFocusItem() ), COleVariant(long(0)), vtPicture
);
  pPicture->Release();
}
```

The following VB.NET sample loads a picture from a file:

```
With AxGrid1.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:

```
axGrid1.Items.set_CellPicture(axGrid1.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.Grid1.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. The CellPictureWidth property has effect on CellPicture property only. 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. The CellPictureWidth property has effect on CellPicture property only. Use the CellPictureHeight property to specify the height of the cell's picture. The CellPictureWidth and CellPictureHeight properties specifies the size of the area where the cell's picture is stretched. If the CellPictureWidth and CellPictureHeight properties are -1 (by default), the cell displays the full size picture. If the CellPictureWidth property is greater than 0, it indicates the width of the area where the cell's picture is stretched.

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

Retrieves or sets a value indicating which radio group a cell is contained in.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Long	A long expression that identifies the cell's radio group.

Use the CellRadioGroup property to add or remove a radio button from a group. In a radio group only one radio button can be checked. A radio cell cannot be contained by two different radio groups. Use the CellHasRadioButton property to add a radio button to a cell. When the cell's state is changed the control fires the CellStateChanged event. The CellState property specifies the cell's state. By default, when a cell of radio type is created the radio cell is not grouped to any of existent radio groups

The following VB sample 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
Grid1.BeginUpdate
With Grid1.Items
For Each h In Grid1.Items
   .CellHasRadioButton(h, 0) = True
   .CellRadioGroup(h, 0) = 1234
Next
End With
Grid1.EndUpdate
```

or

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
   Grid1.Items.CellHasRadioButton(Item, 0) = True
   Grid1.Items.CellRadioGroup(Item, 0) = 1234
End Sub
```

To find out the radio cell that is checked in the radio group 1234 you can use: MsgBox

Grid1.Items.CellValue(, Grid1.Items.CellChecked(1234))

The following sample groups all cells of the first column into a radio group, and displays the checked cell:

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
   Grid1.Items.CellHasRadioButton(Item, 0) = True
   Grid1.Items.CellRadioGroup(Item, 0) = 1234 ' The 1234 is arbirary and it represents the identifier for the radio group
End Sub

Private Sub Grid1_CellStateChanged(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As Long)
   Debug.Print "In the 1234 radio group the """ & Grid1.Items.CellValue(, Grid1.Items.CellChecked(1234)) & """ is checked."
End Sub
```

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

```
With Grid1.Items
   .CellHasRadioButton(.FocusItem, Grid1.FocusColumnIndex) = True
   .CellRadioGroup(.FocusItem, Grid1.FocusColumnIndex) = 1234
End With
```

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

```
#include "Items.h"
CItems items = m_grid.GetItems();
COleVariant vtItem(items.GetFocusItem()), vtColumn( m_grid.GetFocusColumnIndex() );
items.SetCellHasRadioButton( vtItem, vtColumn, TRUE );
items.SetCellRadioGroup( vtItem, vtColumn, 1234 );
```

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

```
With AxGrid1.Items
   .CellHasRadioButton(.FocusItem, AxGrid1.FocusColumnIndex) = True
   .CellRadioGroup(.FocusItem, AxGrid1.FocusColumnIndex) = 1234
End With
```

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

```
axGrid1.Items.set_CellHasRadioButton(axGrid1.Items.FocusItem,
axGrid1.FocusColumnIndex, true);
axGrid1.Items.set_CellRadioGroup(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex,
1234);
```

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

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .CellHasRadioButton(0,thisform.Grid1.FocusColumnIndex) = .t.
   .CellRadioGroup(0,thisform.Grid1.FocusColumnIndex) = 1234
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

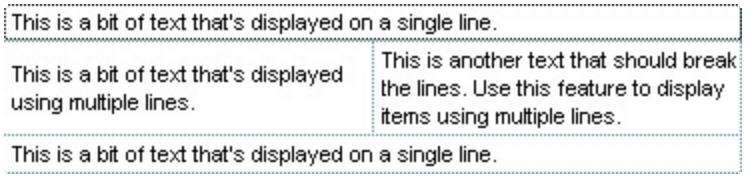
```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

Retrieves or sets a value indicating whether the cell is painted using one line, or more than one line.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key
CellSingleLineEnum	A CellSingleLineEnum expression that indicates whether the cell displays its caption using one or more lines.

By default, the CellSingleLine property is exCaptionSingleLine / True, which indicates that the cell's caption is displayed on a single line. Use the Def(exCellSingleLine) property to specify that all cells in the column display their content using multiple lines. The control can displays the cell's caption using more lines, if the CellSingleLine property is exCaptionWordWrap or exCaptionBreakWrap. The CellSingleLine property wraps the cell's caption so it fits in the cell's client area. If the text doesn't fit the cell's client area, the height of the item is increased or decreased. When the CellSingleLine is exCaptionWordWrap / exCaptionBreakWrap / False, the height of the item is computed based on each cell caption. *If the CellSingleLine property is* exCaptionWordWrap / exCaptionBreakWrap / *False, changing the ItemHeight property has no effect.* Use the ItemMaxHeight property to specify the maximum height of the item when its height is variable. Use the CellVAlignment property to align vertically a cell.



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 Grid1.Items
   .CellSingleLine(.FocusItem, Grid1.FocusColumnIndex) = True
End With
```

The following C++ sample displays the caption of the focused cell using multiple lines:

```
#include "Items.h"
CItems items = m_grid.GetItems();
items.SetCellSingleLine( COleVariant( items.GetFocusItem() ), COleVariant(
long(m_grid.GetFocusColumnIndex()) ), FALSE );
```

The following VB.NET sample displays the caption of the focused cell using multiple lines:

```
With AxGrid1.Items
   .CellSingleLine(.FocusItem, AxGrid1.FocusColumnIndex) = False
End With
```

The following C# sample displays the caption of the focused cell using multiple lines:

```
axGrid1.Items.set_CellSingleLine(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex,
false);
```

The following VFP sample displays the caption of the focused cell using multiple lines:

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .CellSingleLine( 0, thisform.Grid1.FocusColumnIndex ) = .f.
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

Specifies the cell's sort data.

Type	Description
Item as Variant	A long expression that indicates the item's handle
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Variant	A long expression that indicates the cell's sort data.

The CellSortData property specifies the value being sorted if the SortType property is SortCellData or SortCellDataString. Use the CellData property to associate an extra data to a cell. Use the CellValue property to specify the cell's value. Use the CellCaption property to get the string being displayed in the cell.

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

Retrieves or sets the cell's state. Affects only check and radio cells.

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

Use the CellState property to change the cell's state. The CellState property has effect only for check and radio cells. Use the CellHasCheckBox property to assign a check box to a cell. Use the CellHasRadioButton property to add a radio button to a cell. The control fires the CellStateChanged event when user changes the cell's state. Use the 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 Grid1.Items
   .CellHasCheckBox(.FocusItem, Grid.FocusColumnIndex) = True
   .CellState(.FocusItem, Grid.FocusColumnIndex) = 1
End With
```

The following C++ sample adds a check box that's checked to the focused cell:

```
#include "Items.h"
CItems items = m_grid.GetItems();
COleVariant vtItem( items.GetFocusItem() ), vtColumn(
long(m_grid.GetFocusColumnIndex()) );
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 AxGrid1.Items
   .CellHasCheckBox(.FocusItem, AxGrid1.FocusColumnIndex) = True
```

```
    .CellState(.FocusItem, AxGrid1.FocusColumnIndex) = 1
End With
```

The following C# sample adds a check box that's checked to the focused cell:

```
axGrid1.Items.set_CellHasCheckBox(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex,
true);
axGrid1.Items.set_CellState(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex, 1);
```

The following VFP sample adds a check box that's checked to the focused cell:

```
with thisform.Grid1.Items
    .DefaultItem = .FocusItem
    .CellHasCheckBox( 0, thisform.Grid1.FocusColumnIndex ) = .t.
    .CellState( 0,thisform.Grid1.FocusColumnIndex ) = 1
endwith
```

The following sample shows how to change the state for a cell to checked state:
Grid1.Items.CellState(Grid1.Items(0), 0) = 1,

The following sample shows how to change the state for a cell to unchecked state:
Grid1.Items.CellState(Grid1.Items(0), 0) = 0,

The following sample shows how to change the state for a cell to partial checked state:
Grid1.Items.CellState(Grid1.Items(0), 0) = 2

The following sample displays a message when a cell of radio or check type has changed its state:

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
    Grid1.Items.CellHasCheckBox(Item, 0) = True
End Sub

Private Sub Grid1_CellStateChanged(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As
Long)
    Debug.Print "The cell """ & Grid1.Items.CellValue(Item, ColIndex) & """ has changed its
state. The new state is " & IIf(Grid1.Items.CellState(Item, ColIndes) = 0, "Unchecked",
"Checked")
End Sub
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely

represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

Retrieves or sets a value that specifies whether the cell should appear in strikeout.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption.
Boolean	A boolean expression that indicates whether the cell should appear in strikeout.

If the CellStrikeOut property is True, the cell's font is displayed with a horizontal line through it. Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the CellValueFormat property to specify an HTML caption. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample draws a horizontal line through the caption of the cell that has the focus:

```
With Grid1.Items
   .CellStrikeOut(.FocusItem, Grid.FocusColumnIndex) = 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_grid.GetItems();
items.SetCellStrikeOut( COleVariant( items.GetFocusItem() ), COleVariant(
(long)m_grid.GetFocusColumnIndex() ), TRUE );
```

The following C# sample draws a horizontal line through the caption of the cell that has the focus:

```
axGrid1.Items.set_CellStrikeOut(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex,
true);
```

The following VB.NET sample draws a horizontal line through the caption of the cell that has the focus:

```
With AxGrid1.Items
   .CellStrikeOut(.FocusItem, AxGrid1.FocusColumnIndex) = True
End With
```

The following VFP sample draws a horizontal line through the caption of the cell that has the focus:

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .CellStrikeOut(0, thisform.Grid1.FocusColumnIndex ) = .t.
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

Retrieves or sets a value that indicates the cell's tool tip text.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
String	A string expression that indicates the cell's tooltip.

By default, the CellToolTip property is "..." (three dots). If the CellToolTip property is "..." the control displays the cell's caption if it doesn't fit the cell's client area. If the CellToolTip property is different than "...", the control shows a tooltip that displays the CellToolTip value. The control fires the ToolTip event when the column's tooltip is about to be displayed. Use the ToolTipWidth property to specify the width of the tooltip window. 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 ^{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

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* that refers a cell.

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

Retrieves or sets a value that indicates whether the cell is underlined.

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

Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the CellValueFormat 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 Grid1.Items
   .CellUnderline(.FocusItem, Grid1.FocusColumnIndex) = True
End With
```

The following C++ sample underlines the focused cell:

```
#include "Items.h"
CItems items = m_grid.GetItems();
items.SetCellUnderline( COleVariant( items.GetFocusItem() ), COleVariant(
(long)m_grid.GetFocusColumnIndex() ), TRUE );
```

The following C# sample underlines the focused cell:

```
axGrid1.Items.set_CellUnderline(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex,
true);
```

The following VB.NET sample underlines the focused cell:

```
With AxGrid1.Items
```

```
    .CellUnderline(.FocusItem, AxGrid1.FocusColumnIndex) = True
End With
```

The following VFP sample underlines the focused cell:

```
with thisform.Grid1.Items
    .DefaultItem = .FocusItem
    .CellUnderline(0, thisform.Grid1.FocusColumnIndex ) = .t.
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
    .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
    .Items.CellBold(.Items(0), 0) = True
    .Items.CellBold(.Items(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True
    .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

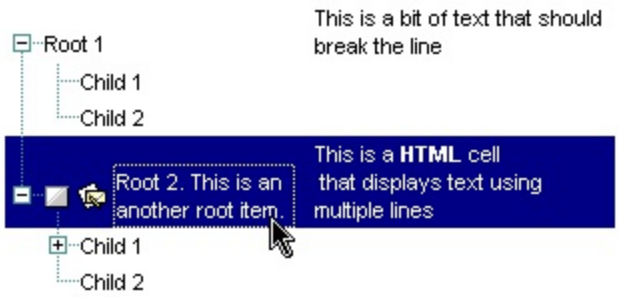
property Items.CellVAlignment ([Item as Variant], [ColIndex as Variant]) as VAlignmentEnum

Retrieves or sets a value that indicates how the cell's caption is vertically aligned.

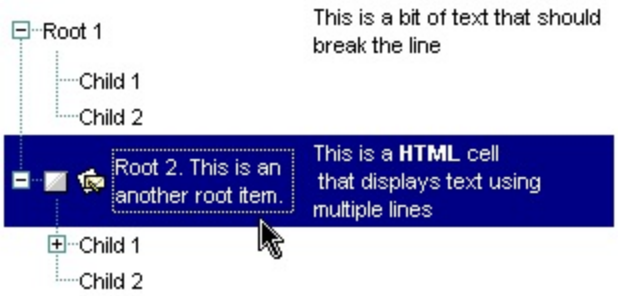
Type	Description
Item as Variant	A long expression that indicates the item's handle
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
VAlignmentEnum	A VAlignmentEnum expression that indicates the cell's vertically alignment.

The CellVAlignment property aligns vertically the cell. The CellVAligment property aligns the +/- sign if the item contains child items. The CellVAlignment property has effect if the item displays cells using multiple lines. Use the CellSingleLine property to wrap the cell's caption on multiple lines. Use the ItemHeight property to specify the height of the item. Use the **
** built-in HTML format to break a line, when CellValueFormat property is exHTML. Use the CellHAlignment property to align horizontally the cell. Use the Def(exCellVAlignment) property to specify the same vertical alignment for the entire column.

The following screen shot shows the "Root 1" and "Root 2" cells that are aligned to the bottom of the item, and the +/- signs are always displayed as CellVAlignment property indicates.



By default, if the CellVAlignmnet property is not used, the sample looks like follows:



The screen shows were generated using the following template:

```
BeginUpdate
LinesAtRoot = -1
MarkSearchColumn = False
Indent = 18

Images("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUlk8plUrlktl



Columns
{
   "Column 1"
   "Column 2"
}
Items
{
   Dim h, h2, h3
   h = AddItem("Root 1")
   CellVAlignment(h,0) = 2
   CellValue(h,1) = "This is a bit of text that should break the line"
   CellSingleLine(h,1) = False
   h2 = InsertItem(h,,"Child 1")
   InsertItem(h,,"Child 2"
   ExpandItem(h) = True
   h = AddItem("Root 2. This is an another root item. ")
   CellImage(h,0) = 1
   CellSingleLine(h,0) = False
   CellHasCheckBox( h,0) = 1
   CellVAlignment(h,0) = 2
   CellValue(h,1) = "This is a HTML cell
 that displays text using
multiple lines"
   CellSingleLine(h,1) = False
   CellValueFormat(h,1) = 1
   ItemHeight(h) = 42
   h2= InsertItem(h,,"Child 1")
   h2= InsertItem(h2,,"Child 1")
   InsertItem(h,,"Child 2")
```

```
    ExpandItem(h) = True
    SelectItem(h) = True
}
EndUpdate
```

Open the control in design mode, select its properties, locate the Template page, and paste the code (the template).

The following VB sample aligns the focused cell to the bottom:

```
With Grid1.Items
    .CellVAlignment(.FocusItem, Grid1.FocusColumnIndex) =
VAlignmentEnum.BottomAlignment
End With
```

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

```
#include "Items.h"
CItems items = m_grid.GetItems();
items.SetCellVAlignment( COleVariant( items.GetFocusItem() ), COleVariant(
(long)m_grid.GetFocusColumnIndex() ), 2 /*BottomAlignment*/ );
```

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

```
With AxGrid1.Items
    .CellVAlignment(.FocusItem, AxGrid1.FocusColumnIndex) =
EXGRIDLib.VAlignmentEnum.BottomAlignment
End With
```

The following C# sample right aligns the focused cell:

```
axGrid1.Items.set_CellVAlignment(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex,
EXGRIDLib.VAlignmentEnum.BottomAlignment);
```

The following VFP sample right aligns the focused cell:

```
with thisform.Grid1.Items
    .DefaultItem = .FocusItem
    .CellVAlignment(0,thisform.Grid1.FocusColumnIndex) = 2 && BottomAlignment
```

```
endwith
```

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

Specifies the cell's value.

Type	Description
Item as Variant	A long expression that indicates the item's handle. During the ValidateValue event, you can uses -1 instead Item, to access to the modified value. In other words during ValidateValue event, the Items.CellValue(Item,ColIndex) and Items.CellCaption(Item,ColIndex) properties retrieve the original value/caption of the cell while the Items.CellValue(-1,ColIndex) and Items.CellCaption(-1,ColIndex) gets the modified value of the specified cell.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key. If the Item parameter is missing or it is zero (0), the ColIndex parameter is the handle of the cell being accessed.
Variant	A variant expression that indicates the cell's value or content. The cell's value supports built-in HTML format if the CellValueFormat property is exHTML.

Use the CellValue property to specify the value or the content for cells in the second, third columns and so on. The CellValueFormat property indicates the way the cell displays its content. The Def(exCellValueFormat) property indicates the format for all cells within the column.

The cell shows its text based on the CellValueFormat property as follows:

- **exText**, the CellValue indicates the text to be displayed without HTML formatting
- **exHTML**, the CellValue indicates the text to be displayed with HTML formatting, such as <b> to bold a portion of text.
- **exComputedField**, the CellValue property indicates a formula to display the cell's content based on the values of any cell in the current item. For instance, the %1 + %2 + %3 adds or concatenates the values from first 3 cells. The exComputedField can be combined with exHTML that indicates that the computed field may display HTML format. The ComputedField property specifies the formula to compute the entire column. The ComputeValue property can be used to get the result of specified formula.
- **exTotalField**, the CellValue indicates a formula to display the cell's content based on the values of any cell from any column, any item or its descendents. For instance, the

sum(1,0,%1 + %2 + %3) gets the sum of first three columns from the direct descendents of the first item. The exTotalField can be combined with exHTML that indicates that the total field may display HTML format. The divider, unsortable or unselectable items do not count for total fields. The ComputeValue property can be used to get the result of specified formula.

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.

The Change event is called when the user changes the CellValue property. Use the CellData property to associate an user data to a cell. The CellSortData property specifies the value being sorted if the SortType property is SortCellData or SortCellDataString. The AddItem or InsertItem method may specify the value for the first cell. Use the LockedItemCount property to lock or unlock items to the top or bottom side of the control. Use the ItemCell property to get the cell's handle based on the item and the column. Use the CellItem property to get the handle of the item that's the owner of the cell. Use the SplitCell property to split a cell. If the CauseValidateValue property is True, the control fires the ValidateValue property when the user changes the CellValue property. Use the AddItem method to add new predefined values to a drop down list editor. Use the CellEditor property to assign an editor to a single cell. Use the Editor property to assign the same editor to all cells in the column. Use the Add method to add new columns to the control. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample displays an HTML cell on multiple lines:

```
With Grid1.Items
    Dim h As HITEM
    h = .AddItem("Cell 1")
    .CellValue(h, 1) = "<r><dotline><b>HTML support</b><br>This is a bit of text
where built-in <b>HTML</b> support is enabled."
    .CellValueFormat(h, 1) = exHTML
    .CellSingleLine(h, 1) = False
    .CellEditorVisible(h, 1) = False
End With
```

The following C++ changes the value of the focused cell:

```
#include "Items.h"
CItems items = m_grid.GetItems();
COleVariant vtItem( items.GetFocusItem() ), vtColumn(
long(m_grid.GetFocusColumnIndex()) );
items.SetCellValue( vtItem, vtColumn, COleVariant("new value") );
```

The following VB.NET changes the value of the focused cell:

```
With AxGrid1.Items
   .CellValue(.FocusItem, AxGrid1.FocusColumnIndex) = "new value"
End With
```

The following C# changes the value of the focused cell:

```
axGrid1.Items.set_CellValue(axGrid1.Items.FocusItem, axGrid1.FocusColumnIndex, "new
value");
```

The following VFP changes the value of the focused cell:

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .CellValue(0,thisform.Grid1.FocusColumnIndex) = "new value"
endwith
```

You may include strings like [m˛], [mł], [180ş], źml, or ˝m˛, žmł, and so on. Copy the symbol from this page, and paste to your cell.

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
```

```
    .Items.CellBold(.Items.ItemByIndex(0)) = True
    .Items.CellBold(.Items.ItemByIndex(0), 0) = True
    .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

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

Specifies how the cell's caption is displayed.

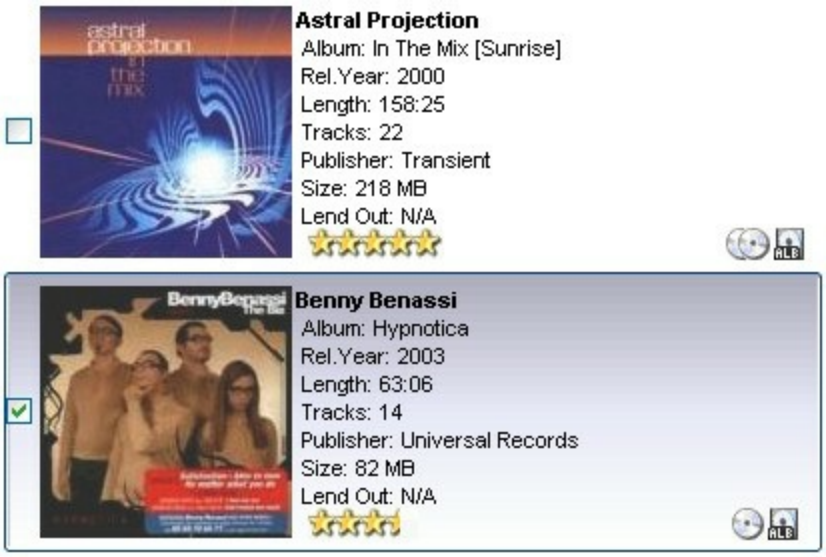
Type	Description
Item as Variant	A long expression that indicates the item's handle
ColIndex as Variant	A long expression that indicates the column's index, or a string expression that indicates the column's caption or column's key.
ValueFormatEnum	A long expression that defines the way how the cell's value is displayed. This value can be an OR combination of listed values. For instance, exHTML + exTotalField indicates a total field that may display HTML format

The component supports built-in HTML format. That means that you can use HTML tags when displays the cell's value . By default, the CellValueFormat property is exText. If the CellValueFormat is exText, the cell displays the CellValue property like it is. If the CellValueFormat is exHTML, the cell displays the CellValue property using the HTML tags specified in the ValueFormatEnum type. Use the Def property to specify whether all cells in the column display HTML format. Use the CellVAlignment property to align vertically a cell.

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.



The following template displays cells using built-in HTML format (the screen shot was generated using the following template) :

```
BeginUpdate
HeaderVisible = False
ScrollBySingleLine = True
BackColor = RGB(255,255,255)
SelBackColor = RGB(240,240,240)
SearchColumnIndex = 2
SelForeCOlor = RGB(0,0,0)
ShowFocusRect = False
MarkSearchColumn = False
TreeColumnIndex = -1
DefaultItemHeight = 140
// Star, HalfOfStar, Disk, Alb
Images("gBJJgBAIEAAGAEGCAAhb/hz/EIAh8Tf5CJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl0



// CheckBox-es
Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0olUrlktl

CheckImage(0) = 5
CheckImage(1) = 6
CheckImage(2) = 7

VisualAppearance
{
```

```
   ' SelectedItem
   Add(4,
"gBFLBCJwBAEHhEJAEGg4BV4Fg6AADACAxRDAMgBQKAAzQFAYZhxBaERiGIZ4JhUAIIRZGN

}
SelBackColor = 67108864            '0x04BBGGRR

Columns
{
   "P"
   {
      Width = 18
      AllowSizing = False
      Def(0) = True
   }
   "Album"
   {
      Width = 126
      AllowSizing = False
   }
   "Description"
   "Album"
   {
      Width = 54
      AllowSizing = False
      Alignment = 1
   }
}
Items
{
   Dim h
   h = AddItem()
'  CellPicture(h,1) = "a1.jpg"
   CellPicture(h,1) =
"gD/bD/cAACBKIxJIwAAIBAEMiAAf7bABDAADAQCjEaAcdAkeAoIAoFAgEAoKA4HAwIBgKB

   CellValue(h,2) = "Astral Projection
```

Album: In The Mix [Sunrise]
Rel.Year: 2000
Length: 158:25
Tracks: 22
Publisher: Transient
Size: 218 MB
Lend Out: N/A
1:131:131:131:131:13"
CellValueFormat(h,2) = 1
CellValue(h,3) = "

3:634"
CellValueFormat(h,3) = 1
h = AddItem()
' CellPicture(h,1) = "a2.jpg"
CellPicture(h,1) =
"gD/bD/cAACBKIxJIwAAIBAEMiAAf7bABDAADAQCjEaAcdAkeAoIAoFAgEAoKA4HAwIBgKB

CellValue(h,2) = "**Benny Benassi**
Album: Hypnotica
Rel.Year: 2003
Length: 63:06
Tracks: 14
Publisher: Universal Records
Size: 82 MB
Lend Out: N/A
1:131:131:132"
CellValueFormat(h,2) = 1
CellValue(h,3) = "

```
34"
   CellValueFormat(h,3) = 1
}
EndUpdate
```

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

Retrieves or sets a value that indicates the width of the inner cell.

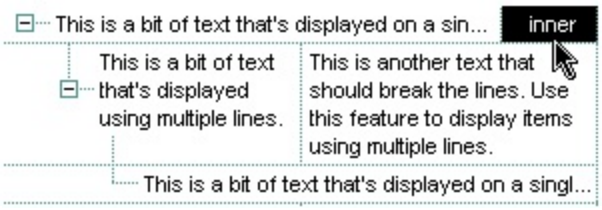
Type	Description
Item as Variant	A long expression that indicates the handle of the item where the cell is, or 0. If the Item parameter is 0, the ColIndex parameter must indicate the handle of the cell.
ColIndex as Variant	A long expression that indicates the index of the column where a cell is divided, 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 specifies that the master cell should have 32 pixels, and the other two inner cells get the same width:

```
With Grid1.Items
  Dim h As HITEM, f As HCELL
```

```
    h = .FirstVisibleItem
    .CellWidth(h, 0) = 32
    f = .ItemCell(h, 0)
    f = .SplitCell(, f)
    f = .SplitCell(, f)
End With
```

The following VB sample specifies that the inner cells should have 32 pixels:

```
With Grid1.Items
    Dim h As HITEM, f As HCELL
    h = .FirstVisibleItem
    f = .SplitCell(h, 0)
    .CellWidth(, f) = 32
End With
```

The following VB sample adds an inner cell to the focused cell with 48 pixels width:

```
Grid1.BeginUpdate
With Grid1.Items
    Dim h As Long
    h = .SplitCell(.FocusItem, 0)
    .CellBackColor(, h) = vbBlack
    .CellForeColor(, h) = vbWhite
    .CellHAlignment(, h) = CenterAlignment
    .CellValue(, h) = "inner"
    .CellWidth(, h) = 48
End With
Grid1.EndUpdate
```

The following C++ sample adds an inner cell to the focused cell with 48 pixels width:

```
#include "Items.h"
m_grid.BeginUpdate();
CItems items = m_grid.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.SetCellValue( vtMissing, vtInner, COleVariant("inner") );
items.SetCellHAlignment( vtMissing, vtInner, 1 );
m_grid.EndUpdate();
```

The following VB.NET sample adds an inner cell to the focused cell with 48 pixels width:

```
With AxGrid1
   .BeginUpdate()
   With .Items
      Dim iInner As Integer
      iInner = .SplitCell(.FocusItem, 0)
      .CellValue(, iInner) = "inner"
      .CellHAlignment(, iInner) = EXGRIDLib.AlignmentEnum.CenterAlignment
      .CellWidth(, iInner) = 48
      .CellBackColor(, iInner) = 0
      .CellForeColor(, iInner) = ToUInt32(Color.White)
   End With
   .EndUpdate()
End With
```

where the ToUInt32 function converts a Color expression to an OLE_COLOR expression and may look like:

```
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 adds an inner cell to the focused cell with 48 pixels width:

```
EXGRIDLib.Items items = axGrid1.Items;
axGrid1.BeginUpdate();
object iInner = items.get_SplitCell(axGrid1.Items.FocusItem, 0);
items.set_CellValue(null, iInner, "inner");
```

```
items.set_CellHAlignment(null, iInner, EXGRIDLib.AlignmentEnum.CenterAlignment);
items.set_CellBackColor(null, iInner, ToUInt32(Color.Black));
items.set_CellForeColor(null, iInner, ToUInt32(Color.White));
items.set_CellWidth(null, iInner, 48);
axGrid1.EndUpdate();
```

where the ToUInt32 function converts a Color to an OLE_COLOR expression and looks like:

```
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);
}
```

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 to count the number of child items. Use the ItemChild property to get the handle of the first child item, if it exists. Use the ItemHasChildren property to built a virtual tree. A virtual tree loads items when the user expands an item. Use the ExpandItem property to expand or collapse an item. Use the InsertItem method to insert child items. Use the InsertControlItem method to insert child ActiveX controls.

method Items.ClearCellBackColor ([Item as Variant], [ColIndex as Variant])

Clears the cell's background color.

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

The ClearCellBackColor method clears the cell's background color when the CellBackColor property is used. Use the ItemBackColor property to specify the item's background color. Use the BackColor property to specify the control's background color

method Items.ClearCellForeColor ([Item as Variant], [ColIndex as Variant])

Clears the cell's foreground color.

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

The ClearCellForeColor method clears the cell's foreground color when CellForeColor property was used. Use the ItemForeColor property to specify the item's foreground color. Use the ForeColor property to specify the control's foreground color.

method Items.ClearCellHAlignment ([Item as Variant], [ColIndex as Variant])

Clears the cell's alignment.

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

Use the ClearCellHAlignment method to clear the alignment of the cell's caption previously set using the CellHAlignment property. If the CellHAlignment property is not called, the Alignment property of the Column object specifies the alignment of the cell's caption.

method Items.ClearItemBackColor (Item as HITEM)

Clears the item's background color.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.

The ClearItemBackColor method clears the item's background color when ItemBackColor property was used. Use the BackColor property to specify the control's background color.

method Items.ClearItemForeColor (Item as HITEM)

Clears the item's foreground color.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.

The ClearItemForeColor method clears the item's foreground color when ItemForeColor property was used. Use the ForeColor property to specify the control's foreground color.

method Items.CollapseAllCards ()

Collapses all the cards.

Type	Description

Use the CollapseAllCards method to collapse all cards in the view. Use the ExpandCard property to programmatically expand or collapse a specified card. Use the ExpandAllCards method to expand all cards in the view. Use the ViewModeOption(exCardViewTitleFormat) property to specify the arrangement of the fields in the title of the cards. Use the HasButtons property to specify whether the control displays an expand/collapse button in the title of the card. Use the ExpandOnKeys property to allow expanding or collapsing the cards using the + or - keys on the numeric keypad. The ExpandOnDblClick property specifies whether a card is expanded or collapsed when a card is double clicked.

property Items.ComputeValue ([Expression as Variant], [Item as Variant], [ColIndex as Variant], [ValueFormatType as Variant]) as Variant

Computes the value of a specified formula.

Type	Description
Expression as Variant	A string expression that specifies the formula to compute
Item as Variant	A long expression that specifies 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.
ValueFormatType as Variant	A ValueFormatType expression that indicates the type of the formula being interpreted by the Expression parameter. For instance, if the ValueFormatType parameter is exTotalField, the Expression parameter should inidcate a total formula of type aggregate(list,direction,formula)
Variant	A string expression that indicates the result.

The ComputeValue property gets the result of a a computed or total field. The Item and ColIndex property refers the cells used as the source for the formula. Use the ComputeValue property to get the result of a total field. For instance, for a total field, the CellValue property indicates the formula, while the ComputeValue can be used to get the result of the formula at runtime.

The ComputeValue method returns the:

- value of the computed field, where the ValueFormatType is exComputedField, and the Expression indicates the formula for the computed field.
- value of the total field, where the ValueFormatType is exTotalField, and the Expression indicates a string as: aggregate(list,direction,formula)
- text with no HTML formatting, where the ValueFormatType is exHTML, and the Expression indicates the string including the HTML format.

For instance, based on the ValueFormatType and Expression parameters the result could be:

- exComputedField, dbl(%0) + dbl(%1), the sum between first two cells in the item referred by Item.
- exTotalField, sum(current,dir,dbl(%0) + dbl(%1)), the total of first two columns, for all direct child items of the item being referred by Item.
- exHTML, bold, returns bold (returns the result with no HTML formatting). In

this case, the Item and ColIndex have no effect.

property Items.DefaultItem as HITEM

Retrieves or sets a value that indicates the handle of the item used by Items properties in VFP.

Type	Description
HITEM	Retrieves the handle of the item that's used by all properties of Items object, that have a parameter Item.

The property is used in VFP implementation. The VFP fires "Invalid Subscript Range" error, while it tries to process a number greater than 65000. Since, the HITEM is a long value that most of the time exceeds 65000, the VFP users have to use this property, instead passing directly the handles to properties. The following sample shows to change the cell's image:

```
.Items.DefaultItem = .Items.AddItem("Item 1")
.Items.CellImage(0,1) = 2
```

In VFP the following sample fires: "Invalid Subscript Range":

```
i = .Items.AddItem("Item 1")
.Items.CellImage(i,1) = 2
```

because the i variable is grater than 65000.

So, if you pass zero to a property that has a parameter titled Item, the control takes instead the DefaultItem value.

method Items.DeleteCellEditor ([Item as Variant], [ColIndex as Variant])

Deletes the cell's built-in editor created by CellEditor property.

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, a string expression that indicates the column's caption or key.

Use the DeleteCellEditor method to delete the editor created using the CellEditor property. Use the CellEditorVisible property to hide or show the cell's editor. Use the HasCellEditor property to check whether the cell contains an editor (being created using the CellEditor property). The DeleteCellEditor method has no effect if the cell contains an editor assigned using the the Editor property of the Column object, or the cell has no editor.

property Items.EnableItem(Item as HITEM) as Boolean

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

Type	Description
Item as HITEM	A long expression that indicates the item's handle
Boolean	A boolean expression that indicates whether the item is enabled or disabled.

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.EndBlockUndoRedo ()

Ends recording the UI operations and adds the undo/redo operations as a block, so they all can be restored at once, if Undo method is performed.

Type	Description

The StartBlockUndoRedo method starts recording the UI operations as a block on undo/redo operations (equivalent of EndBlockUndoRedo method of the control). The method has effect only if the AllowUndoRedo property is True. The EndBlockUndoRedo method collects all undo/redo operations since StartBlockUndoRedo method was called and add them to the undo/redo queue as a block. This way the next call on a Undo operation, the entire block is restored, so all UI operations are restored. The EndBlockUndoRedo method must be called the same number of times as the StartBlockUndoRedo method was called. For instance, if you have called the StartBlockUndoRedo twice the EndBlockUndoRedo method must be called twice too, and the collected operations are added to the control's queue of undo/redo operations at the end.

method Items.EnsureVisibleItem (Item as HITEM)

Ensures that the given item is in the visible client area.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.

The EnsureVisibleItem scrolls the control's content until the item fits the visible client area. The EnsureVisibleItem method expands the parent items. Use the IsItemVisible property to check if an item fits the control's client area. Use the EnsureVisibleColumn method to scroll the control's content so a column fits the control's client area. Use the Scroll method to scroll the control's client area by code. The EnsureVisibleItem method should not be called during BeginUpdate and EndUpdate methods. The EnsureOnSort property prevents scrolling the control's content when the user sorts items.

The following VB sample ensures that the last added item fits the control's client area:

```
With Grid1.Items
  .EnsureVisibleItem .ItemByIndex(.ItemCount - 1)
End With
```

The following C++ sample ensures that the last added item fits the control's client area:

```
#include "Items.h"
m_grid.BeginUpdate();
CItems items = m_grid.GetItems();
items.EnsureVisibleItem( items.GetItemByIndex( items.GetItemCount() - 1 ) );
```

The following VB.NET sample ensures that the last added item fits the control's client area:

```
With AxGrid1.Items
    .EnsureVisibleItem(.ItemByIndex(.ItemCount - 1))
End With
```

The following C# sample ensures that the last added item fits the control's client area:

```
axGrid1.Items.EnsureVisibleItem(axGrid1.Items[axGrid1.Items.ItemCount - 1]);
```

The following VFP sample ensures that the last added item fits the control's client area:

```
with thisform.Grid1.Items
```

```
    .EnsureVisibleItem(.ItemByIndex(.ItemCount-1))
endwith
```

method Items.ExpandAllCards ()

Expands all the cards.

Type	Description

Use the ExpandAllCards method to expand all cards in the view. Use the ExpandCard property to programmatically expand or collapse a specified card. Use the CollapseAllCards method to collapse all cards in the view. Use the ViewModeOption(exCardViewTitleFormat) property to specify the arrangement of the fields in the title of the cards. Use the HasButtons property to specify whether the control displays an expand/collapse button in the title of the card. Use the ExpandOnKeys property to allow expanding or collapsing the cards using the + or - keys on the numeric keypad. The ExpandOnDblClick property specifies whether a card is expanded or collapsed when a card is double click

property Items.ExpandCard(Item as HITEM) as Boolean

Expands or collapses the card.

Type	Description
Item as HITEM	A long expression that indicates the handle of the card being expanded or collapsed.
Boolean	A boolean expression that indicates whether the card is expanded or collapsed.

Use the ExpandCard property to programmatically expand or collapse a specified card. Use the ExpandAllCards method to expand all cards in the view. Use the CollapseAllCards method to collapse all cards in the view. Use the ViewModeOption(exCardViewTitleFormat) property to specify the arrangement of the fields in the title of the cards. Use the HasButtons property to specify whether the control displays an expand/collapse button in the title of the card. Use the ExpandOnKeys property to allow expanding or collapsing the cards using the + or - keys on the numeric keypad. The ExpandOnDblClick property specifies whether a card is expanded or collapsed when a card is double clicked.

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.
Boolean	A boolean expression that indicates whether the item is expanded or collapsed.

Use ExpandItem property to programmatically expand or collapse an item, in TableView mode. 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:
Grid1.Items.ExpandItem(Grid1.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. In CardView mode, use the ExpandCard property to expand or collapse a card.

The following VB sample expands the selected item:
Grid1.Items.ExpandItem(Grid1.Items.SelectedItem()) = True**.**

The ExpandItem property has no effect if the item has no child items. Use ChildCount property to determine whether an item contains child nodes. Use the ItemHasChildren property to built a virtual grid. A virtual grid loads items when the the user expands an item.

The following VB sample expands the selected item:

```
Private Sub Grid1_SelectionChanged()
   Grid1.Items.ExpandItem(Grid1.Items.SelectedItem()) = True
End Sub
```

The following VB sample expands programmatically the focused item:

```
With Grid1.Items
```

```
    .ExpandItem(.FocusItem) = True
End With
```

The following C++ sample expands programmatically the focused item:

```
#include "Items.h"
CItems items = m_grid.GetItems();
items.SetExpandItem( items.GetFocusItem(), TRUE );
```

The following VB.NET sample expands programmatically the focused item:

```
AxGrid1.Items.ExpandItem( AxGrid1.Items.FocusItem ) = True
```

The following C# sample expands programmatically the focused item:

```
axGrid1.Items.set_ExpandItem( axGrid1.Items.FocusItem, true );
```

The following VFP sample expands programmatically the focused item:

```
with thisform.Grid1.Items
    .DefaultItem = .FocusItem
    .ExpandItem( 0 ) = .t.
endwith
```

property Items.FindItem (Value as Variant, [ColIndex as Variant], [StartIndex as Variant]) as HITEM

Finds an item, looking for Caption in ColIndex colum. The searching starts at StartIndex item.

Type	Description
Value as Variant	A Variant expression that indicates the caption that is searched for.
ColIndex as Variant	A string expression that indicates the column's caption, or a long expression that indicates the column's index.
StartIndex as Variant	A long value that indicates the index of item from where the searching starts.
HITEM	A long expression that indicates the item's handle that matches the criteria.

Use the FindItem to search for an item. Finds a control's item that matches CellValue(Item, ColIndex) = Caption. The StartIndex parameter indicates the index from where the searching starts. If it is missing, the searching starts from the item with the 0 index. The searching is case sensitive only if the ASCIIUpper property is empty. Use the AutoSearch property to enable incremental search feature within the column.

The following VB sample selects the first item that matches "DUMON" on the first column:

```
Grid1.Items.SelectItem(Grid1.Items.FindItem("DUMON", 0)) = True
```

The following C++ sample finds and selects an item:

```
#include "Items.h"
CItems items = m_grid.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:

```
axGrid1.Items.set_SelectItem(axGrid1.Items.get_FindItem("Child 2", 0, 0), true);
```

The following VB.NET sample finds and selects an item:

```
With AxGrid1.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.Grid1.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 value that indicates the value being searched
StartIndex as Variant	A long expression that indicates the handle 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. 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:

```
Grid1.Items.SelectItem(Grid1.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_grid.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 AxGrid1.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 = axGrid1.Items.get_FindPath("Files and Folders\\Hidden Files and Folders\\Do
not show hidden files and folder");
if ( iFind != 0 )
   axGrid1.Items.set_SelectItem(iFind, true);
```

The following VFP sample selects the item based on its path:

```
with thisform.Grid1.Items
   .DefaultItem = .FindPath("Files and Folders\Hidden Files and Folders\Do not show
hidden files and folder")
   if ( .DefaultItem <> 0 )
      .SelectItem( 0 ) = .t.
   endif
endwith
```

property Items.FirstVisibleItem as HITEM

Retrieves the handle of the first visible item in control.

Type	Description
HITEM	A long expression that indicates the item's handle that indicates 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. Use the RootItem property to get the first visible item in the list. The NextSiblingItem property retrieves the next sibling of the item in the parent's child list. Use the PrevVisibleItem property to retrieve the previous visible item.

The following VB sample enumerates the items that fit the control's client area:

```
Dim h As HITEM
Dim i As Long, j As Long, nCols As Long
nCols = Grid1.Columns.Count
With Grid1.Items
   h = .FirstVisibleItem
   While Not (h = 0) And .IsItemVisible(h)
      Dim s As String
      s = ""
      For j = 0 To nCols - 1
         s = s + .CellValue(h, j) + Chr(9)
      Next
      Debug.Print s
      h = .NextVisibleItem(h)
   Wend
End With
```

The following VB sample enumerates the visible items in the control as they are displayed (sorted):

```
With Grid1.Items
   Dim h As HITEM
   h = .RootItem(0)
   While Not h = 0
```

```
        Debug.Print .CellValue(h, 0)
        h = .NextVisibleItem(h)
    Wend
End With
```

The following VB sample enumerates the items in the control as they are displayed (sorted):. For instance, the sample lists the child items of items that are collapsed too:

```
 With Grid1.Items
    Dim h As HITEM
    h = .RootItem(0)
    While Not h = 0
        Debug.Print .CellValue(h, 0)
        h = .NextSiblingItem(h)
    Wend
End With
```

The following C++ sample enumerates the items that fit the control's client area:

```
#include "Items.h"
CItems items = m_grid.GetItems();
long hItem = items.GetFirstVisibleItem();
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
while ( hItem && items.GetIsItemVisible( hItem, vtMissing ) )
{
    OutputDebugString( V2S( &items.GetCellValue( COleVariant( hItem ), COleVariant(
long(0) ) ) ) );
    hItem = items.GetNextVisibleItem( hItem );
}
```

where the V2S function converts a VARIANT value to a string expression and looks like:

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;
```

```
        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}
```

The following VB.NET sample enumerates the items that fit the control's client area:

```
With AxGrid1.Items
    Dim hItem As Integer = .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:

```
EXGRIDLib.Items items = axGrid1.Items;
int hItem = items.FirstVisibleItem;
while ((hItem != 0) && (items.get_IsItemVisible(hItem, null)))
{
    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.Grid1.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 item's handle that is focused.

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 FocusColumnIndex property to change the focused column. The SelectColumnInner property indicates the index of an inner cell that has the focus. Use the Edit method to start editing the focused cell. Select a new item to focus a new item. Use the SelectItem property to select a new item. Use the Edit method to edit the focused cell, if the AutoEdit property is False. If the control supports single selection, the FocusItem property gets the selected item too. Use the SingleSel property to specify whether the control supports single or multiple selection. 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.

The control fires the FocusChanged event when the user changes:

- the focused item
- the focused column or an inner cell gets the focus.

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 CellValue property indicates the cell's value.

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):

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

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

The expression supports cell's identifiers as follows:

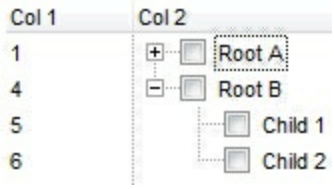
- ***%0**, **%1**, **%2**, ... specifies the value of the cell in the column with the index 0, 1 2, ... The CellValue property specifies the cell's value. For instance, "%0 format ``" formats the value on the cell with the index 0, using current regional setting, while "int(%1)" converts the value of the column with the index 1, to integer.*
- ***%C0**, **%C1**, **%C2**, ... specifies the caption of the cell, or the string the cell displays in the column with the index 0, 1 2, ... The CellCaption property specifies the cell's caption. The cell's value may be different than what the cell displays as a string. For instance, let's say a cell display HTML format. The %0 returns the html format including the HTML tags, while %C0 returns the cell's content as string without HTML tags. For instance, "upper(%C1)" converts the caption of the cell with the index 1, to upper case, while "%C0 left 2" returns the leftmost two characters on the cell with the index 0.*
- ***%CD0**, **%CD1**, **%CD2**, ... 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 instance, "%CD0 = `your user data`" specifies all cells whose CellData property is `your user data`, on the column with the index 0.*
- ***%CS0**, **%CS1**, **%CS2**, ... specifies the cell's state in the column with the index 0, 1 2, ... The CellState property specifies the cell's state, and so it indicates whether the cell is checked or un-checked. For instance, "%CS0" defines all checked items on the column with the index 0, or "not %CS1" defines all un-checked items in the column with the index 1.*

The predefined operators for auto-numbering are:

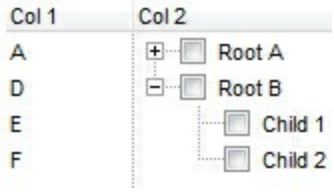
- number **index** 'format', indicates the index of the item. The first added item has the index 0, the second added item has the index 1, and so on. The index of the item remains the same even if the order of the items is changed by sorting. For instance, 1 index '' gets the index of the item starting 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 ""

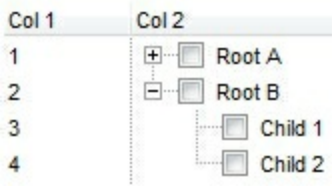


In the following screen shot the FormatColumn("Col 1") = "1 index 'A-Z'"

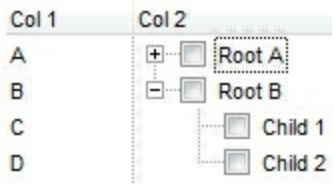


- number **apos** 'format' indicates the absolute position of the item. The first displayed item has the absolute position 0 (scrolling position on top), the next visible item is 1, and so on. The number indicates the starting position, while the format is a set of characters to be used for specifying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

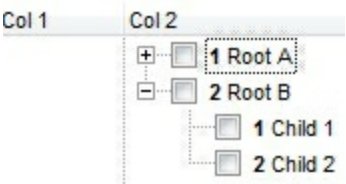


  In the following screen shot the FormatColumn("Col 1") = "1 apos 'A-Z'"

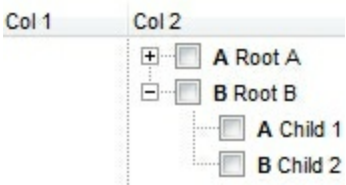


- number **pos** 'format' indicates the relative position of the item. The relative position is the position of the visible child item in the parent children collection. The number indicates the starting position, while the format is a set of characters to be used for specifying the position. For instance,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



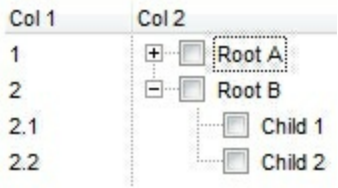
  In the following screen shot the FormatColumn("Col 2") = "'' + 1 pos 'A-Z' + ' ' + value"



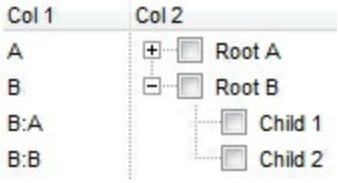
- number **opos** 'format' indicates the relative old position of the item. The relative old position is the position of the child item in the parent children collection. The number indicates the starting position, while the format is a set of characters to be used for specifying the position.For instance,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

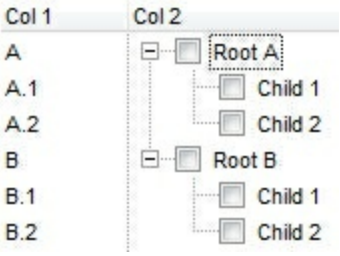
In the following screen shot the FormatColumn("Col 1") = "1 rpos ""



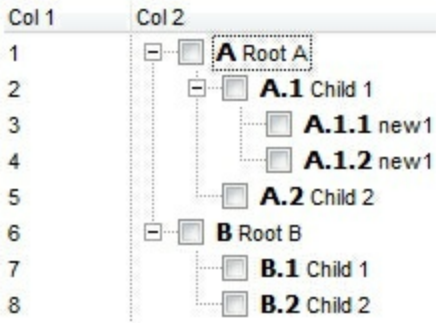
In the following screen shot the FormatColumn("Col 1") = "1 rpos ':|A-Z'"



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 an ExGrid 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 CellValue property values of all its ancestors.

property Items.GroupItem (Item as HITEM) as Long

Indicates a group item if positive, and the value specifies the index of the column that has been grouped.

Type	Description
Item as HITEM	A Long expression that specifies the handle of the item being queried
Long	A Long expression that specifies index of the column being grouped, or a negative value if the item is a regular item, not a grouping item.

The GroupItem method determines the index of the column that indicates the column being grouped. In other words, the CellCaption(Item,GroupItem(Item)) gets the default caption to be displayed for the grouping item. The Ungroup method removes all grouping items. For instance, when a column gets grouped by, the control sorts by that column, collects the unique values being found, and add a new item for each value found, by adding the items of the same value as children. The (AddGroupItem event is fired for each new item to be inserted in the Items collection during the grouping.

The following samples show how to display the grouping items with a solid background color, instead of a single line:

VBA

```
Private Sub Grid1_AddGroupItem(ByVal Item As Long)
  With Grid1
    With .Items
      .ItemDividerLine(Item) = 0
      .CellHAlignment(Item,.GroupItem(Item)) = 1
      .ItemBackColor(Item) = RGB(240,240,240)
    End With
  End With
End Sub
```

VB

```
Private Sub Grid1_AddGroupItem(ByVal Item As EXGRIDLibCtl.HITEM)
  With Grid1
    With .Items
      .ItemDividerLine(Item) = EmptyLine
```

```
        .CellHAlignment(Item,.GroupItem(Item)) = CenterAlignment
        .ItemBackColor(Item) = RGB(240,240,240)
    End With
  End With
End Sub
```

VB.NET

```
Private Sub Exgrid1_AddGroupItem(ByVal sender As System.Object,ByVal Item As Integer)
Handles Exgrid1.AddGroupItem
  With Exgrid1
    With .Items
        .set_ItemDividerLine(Item,exontrol.EXGRIDLib.DividerLineEnum.EmptyLine)

.set_CellHAlignment(Item,.get_GroupItem(Item),exontrol.EXGRIDLib.AlignmentEnum.Center/

        .set_ItemBackColor(Item,Color.FromArgb(240,240,240))
    End With
  End With
End Sub
```

C++

```
void OnAddGroupItemGrid1(long Item)
{
  EXGRIDLib::IGridPtr spGrid1 = GetDlgItem(IDC_GRID1)->GetControlUnknown();
  EXGRIDLib::IItemsPtr var_Items = spGrid1->GetItems();
    var_Items->PutItemDividerLine(Item,EXGRIDLib::EmptyLine);
    var_Items->PutCellHAlignment(Item,var_Items-
>GetGroupItem(Item),EXGRIDLib::CenterAlignment);
    var_Items->PutItemBackColor(Item,RGB(240,240,240));
}
```

C++ Builder

```
void __fastcall TForm1::Grid1AddGroupItem(TObject *Sender,Exgridlib_tlb::HITEM Item)
{
  Exgridlib_tlb::IItemsPtr var_Items = Grid1->Items;
```

```
    var_Items->set_ItemDividerLine(Item,Exgridlib_tlb::DividerLineEnum::EmptyLine);
    var_Items->set_CellHAlignment(TVariant(Item),TVariant(var_Items-
>get_GroupItem(Item)),Exgridlib_tlb::AlignmentEnum::CenterAlignment);
    var_Items->set_ItemBackColor(Item,RGB(240,240,240));
}
```

C#

```
private void exgrid1_AddGroupItem(object sender,int Item)
{
   exontrol.EXGRIDLib.Items var_Items = exgrid1.Items;
    var_Items.set_ItemDividerLine(Item,exontrol.EXGRIDLib.DividerLineEnum.EmptyLine);

var_Items.set_CellHAlignment(Item,var_Items.get_GroupItem(Item),exontrol.EXGRIDLib.Alig

    var_Items.set_ItemBackColor(Item,Color.FromArgb(240,240,240));
}
```

JavaScript

```
<SCRIPT FOR="Grid1" EVENT="AddGroupItem(Item)" LANGUAGE="JScript">
   var var_Items = Grid1.Items;
    var_Items.ItemDividerLine(Item) = 0;
    var_Items.CellHAlignment(Item,var_Items.GroupItem(Item)) = 1;
    var_Items.ItemBackColor(Item) = 15790320;
</SCRIPT>
```

X++ (Dynamics Ax 2009)

```
void onEvent_AddGroupItem(int _Item)
{
   COM com_Items;
   anytype var_Items;
   ;
   var_Items = exgrid1.Items(); com_Items = var_Items;
    com_Items.ItemDividerLine(_Item,0/*EmptyLine*/);

com_Items.CellHAlignment(_Item,com_Items.GroupItem(_Item),1/*CenterAlignment*/);
```

```
    com_Items.ItemBackColor(_Item,WinApi::RGB2int(240,240,240));
}
```

VFP

```
*** AddGroupItem event - Occurs after a new Group Item has been inserted to Items
collection. ***
LPARAMETERS Item
   with thisform.Grid1
      with .Items
         .ItemDividerLine(Item) = 0
         .CellHAlignment(Item,.GroupItem(Item)) = 1
         .ItemBackColor(Item) = RGB(240,240,240)
      endwith
   endwith

with thisform.Grid1
   .BeginUpdate
   .HasLines = 0
   .ColumnAutoResize = .F.
   rs = CreateObject("ADOR.Recordset")
   with rs
      var_s = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program
Files\Exontrol\ExGrid\Sample\SAMPLE.MDB"
      .Open("Orders",var_s,3,3)
   endwith
   .DataSource = rs
   .SingleSort = .F.
   .SortBarVisible = .T.
   .AllowGroupBy = .T.
   .Columns.Item(1).SortOrder = .T. && .T.
   .EndUpdate
endwith
```

Delphi (standard)

```
procedure TForm1.Grid1AddGroupItem(ASender: TObject; Item : HITEM);
begin
```

```
  with Grid1 do
  begin
    with Items do
    begin
      ItemDividerLine[Item] := EXGRIDLib_TLB.EmptyLine;
      CellHAlignment[OleVariant(Item),OleVariant(GroupItem[Item])] :=
EXGRIDLib_TLB.CenterAlignment;
      ItemBackColor[Item] := $f0f0f0;
    end;
  end
end;
```

Visual Objects

```
METHOD OCX_Exontrol1AddGroupItem(Item) CLASS MainDialog
  // AddGroupItem event - Occurs after a new Group Item has been inserted to
Items collection.
  local var_Items as IItems
  var_Items := oDCOCX_Exontrol1:Items
    var_Items:[ItemDividerLine,Item] := EmptyLine
    var_Items:[CellHAlignment,Item,var_Items:[GroupItem,Item]] := CenterAlignment
    var_Items:[ItemBackColor,Item] := RGB(240,240,240)
RETURN NIL
```

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

Specifies whether a cell has a built-in editor.

Type	Description
Item as Variant	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the cell's handle or the column's index, a string expression that indicates the column's caption or the column's key.
Boolean	A boolean expression that indicates whether the cell has a built-in editor created using the CellEditor method.

Use the HasCellEditor property to check whether the cell has an individual editor being added using the CellEditor method before. Use the HasCellEditor property to find if a cell has a particular editor. The HasCellEditor property gets true only if the cell has its own editor assigned, it never gets true, if the cell's column has an editor. Use the CellEditor method to assign different editors in the same column. Use the Editor property to assign the same editor for all cells in the column.

The following VB sample shows the drop down portion of the control when a cell is focused:

```
Private Sub Grid1_FocusChanged()
  With Grid1
    Dim i As Long
    i = .FocusColumnIndex
    With Grid1.Items
      If (.CellEditorVisible(.FocusItem, i)) Then
        Dim e As EXGRIDLibCtl.Editor
        Set e = Grid1.Columns(i).Editor
        If .HasCellEditor(.FocusItem, i) Then
          Set e = .CellEditor(.FocusItem, i)
        End If
        If Not e Is Nothing Then
          e.DropDown
        End If
      End If
    End With
  End With
End Sub
```

The following VB sample assigns a date type editor to the focused cell (the sample checks first if the cell doesn't have already an editor):

```
With Grid1.Items
   Dim h As EXGRIDLibCtl.HITEM
   h = .FocusItem
   If Not .HasCellEditor(h, Grid1.FocusColumnIndex) Then
      With .CellEditor(h, Grid1.FocusColumnIndex)
         .EditType = DateType
      End With
   End If
End With
```

The following C++ sample assigns a date type editor to the focused cell (the sample checks first if the cell doesn't have already an editor):

```
#include "Items.h"
#include "Editor.h"
CItems items = m_grid.GetItems();
COleVariant vtItem( items.GetFocusItem() ), vtColumn(
long(m_grid.GetFocusColumnIndex() ) );
if ( !items.GetHasCellEditor( vtItem, vtColumn ) )
{
   CEditor editor = items.GetCellEditor( vtItem, vtColumn );
   editor.SetEditType( 7 /*DateType*/ );
}
```

The following VB.NET sample assigns a date type editor to the focused cell (the sample checks first if the cell doesn't have already an editor):

```
With AxGrid1.Items
   Dim hItem As Integer = .FocusItem
   If Not .HasCellEditor(hItem, AxGrid1.FocusColumnIndex) Then
      With .CellEditor(hItem, AxGrid1.FocusColumnIndex)
         .EditType = EXGRIDLib.EditTypeEnum.DateType
      End With
   End If
End With
```

The following C# sample assigns a date type editor to the focused cell (the sample checks first if the cell doesn't have already an editor):

```
EXGRIDLib.Items items = axGrid1.Items;
int hItem = items.FocusItem;
if (hItem != null)
   if (!items.get_HasCellEditor(hItem, axGrid1.FocusColumnIndex))
   {
      EXGRIDLib.Editor editor = items.get_CellEditor(hItem, axGrid1.FocusColumnIndex);
      editor.EditType = EXGRIDLib.EditTypeEnum.DateType;
   }
```

The following VFP sample assigns a date type editor to the focused cell (the sample checks first if the cell doesn't have already an editor):

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   if ( !.HasCellEditor(0, thisform.Grid1.FocusColumnIndex ) )
      with .CellEditor( 0, thisform.Grid1.FocusColumnIndex )
         .EditType = 7 && DateType
      endwith
   endif
endwith
```

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 splitted cells, the InnerCell(, , 3), or InnerCell(, , 4), and so on, always retrievs 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 EXGRIDLibCtl.Grid, ByVal h As EXGRIDLibCtl.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 EXGRIDLibCtl.Grid, ByVal h As EXGRIDLibCtl.HITEM,
ByVal c As Long) As EXGRIDLibCtl.HCELL
  With g.Items
    Dim r As EXGRIDLibCtl.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 VB following sample enumerates the list of the inner cells (including the cell where the splitting starts):

```
Private Sub enumSplit(ByVal g As EXGRIDLibCtl.Grid, ByVal h As EXGRIDLibCtl.HITEM,
ByVal c As Long)
  With g.Items
    Dim i As Long
    i = -1
    Do
      i = i + 1
      Debug.Print .CellCaption(, .InnerCell(h, c, i))
    Loop While Not (.InnerCell(h, c, i) = .InnerCell(h, c, i + 1))
  End With
End Sub
```

The VB following sample enumerates the list of inner cells, starting from the master cell:

```
enumSplit Grid1, 0, getMaster(Grid1, h, c)
```

The following VB sample counts the inner cells:

```
Private Function getInnerCount(ByVal g As EXGRIDLibCtl.Grid, ByVal h As
EXGRIDLibCtl.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 VC 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( CGrid& grid, long h, long c )
{
  CItems items = grid.GetItems();
  return V2I( &items.GetInnerCell( COleVariant( h ), COleVariant( c ), COleVariant( (long)0 )
) ) != V2I( &items.GetInnerCell( COleVariant( h ), COleVariant( c ), COleVariant( (long)1 ) ) );
}
```

The following VC sample gets the master cell:

```
long getMaster( CGrid& grid, long h, long c )
{
  COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
  CItems items = grid.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 VC sample counts the inner cells:

```
long getInnerCount( CGrid& grid, long h, long c )
{
   CItems items = grid.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 AxGrid1.Items
   Dim i As Object
   i = .SplitCell(.FirstVisibleItem, 0)
   .CellValue(Nothing, i) = "inner cell"
End With
```

The following C# sample splits the first visible cell in two cells:

```
EXGRIDLib.Items items = axGrid1.Items;
object i = items.get_SplitCell(items.FirstVisibleItem, 0);
items.set_CellValue(null, i, "inner cell");
```

The following VFP sample splits the first visible cell in two cells:

```
with thisform.Grid1.Items
   local i
   i = .SplitCell(.FirstVisibleItem,0)
   local s, crlf
   crlf = chr(13) + chr(10)
```

```
    s = "Items" + crlf
    s = s + "{" + crlf
    s = s + "CellValue(," + str(i) + ") = " + chr(34) + "inner cell" + chr(34) + crlf
    s = s + "}"
    thisform.Grid1.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 for the component being inserted, if required. Only, the vendor of the component you are going to use is able to give you such of runtime license, so please contact the control's vendor for such of key. Your development license key is not compatible with the runtime license key, so it can't be used here.

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

The control supports ActiveX hosting, so you can insert any ActiveX component as a child item of the control. If you are using the /NET assembly you can use the InsertObjectItem property to insert a /NET control as a child item of the control. The InsertControlItem property creates the specified ActiveX control and hosts to a new child item of the control, while the InsertObjectItem property hosts the already created object to a new child item of the control. An inner control sends notifications/events to parent control through the ItemOleEvent event.

The ControlID must be formatted in one of the following ways:

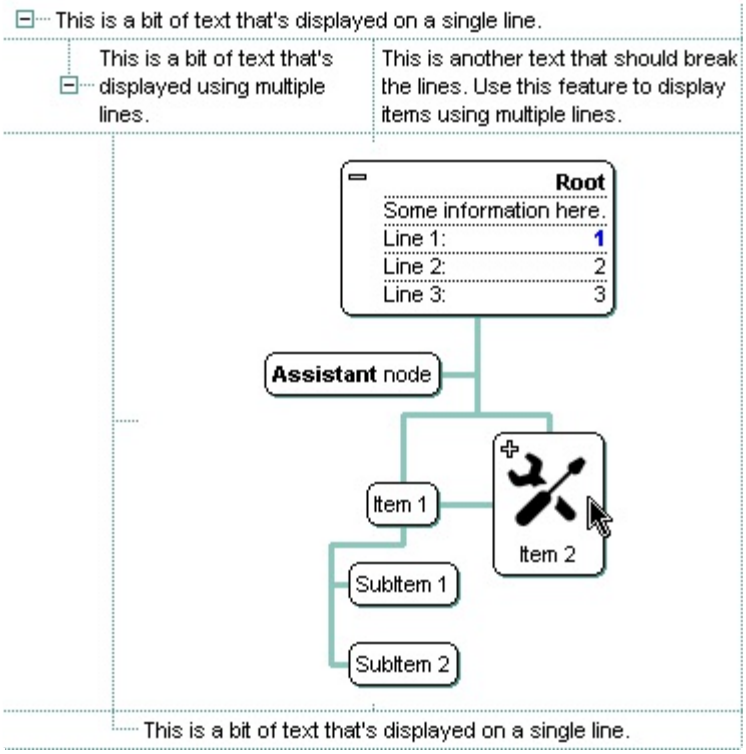
- A ProgID such as "Exontrol.Grid"
- A CLSID such as "{8E27C92B-1264-101C-8A2F-040224009C02}"
- A URL such as "https://www.exontrol.com"

- A reference to an Active document such as "c:\temp\myfile.doc", or "c:\temp\picture.gif"
- A fragment of HTML such as "MSHTML:<HTML><BODY>This is a line of text</BODY></HTML>"
- A fragment of XML

In case the control you want to insert fails, you can add the "A2X:" prefix to the ControlID such as:

- A ProgID such as "A2X:Exontrol.Grid"
- A CLSID such as "A2X:{8E27C92B-1264-101C-8A2F-040224009C02}"
- A URL such as "A2X:https://www.exontrol.com"
- A reference to an Active document such as "A2X:c:\temp\myfile.doc", or "c:\temp\picture.gif"
- A fragment of HTML such as "A2X:MSHTML:<HTML><BODY>This is a line of text</BODY></HTML>"

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 exGrid 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. Use the ItemAllowSizing property to let user resizes the item at runtime, and so the object being hosted. 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.

You can use one of the following methods to find out information about the inner ActiveX control:

- Looking for the inner ActiveX control's documentation. For instance, if you are inserting a "MSCAL.Calendar" control you need to know how to use and run the "Microsoft Calendar Control" in order to call its properties or methods.
- Using the OLE View tool (that's installed by MSDEV) to list the inner ActiveX control's type library.
- Using the Exontrol's exPropertiesList control to browse the inner ActiveX control properties at runtime. Create a new project, insert an exGrid and exPropertiesList control inside the main form, and use the following snippet of code:

```
Private Sub Form_Load()
   With Grid1
      .BeginUpdate
      .Columns.Add "Column 1"
      .Items.InsertControlItem , "Exontrol.ChartView"
      .EndUpdate
      With .Items
         PropertiesList1.Select .ItemObject(.ItemByIndex(0))
      End With
   End With
End Sub
```

  This way the exPropertiesList control browses the object that's hosted by the first item in the exGrid control, so you will be able to see the properties that you will be able to call them for the inner ActiveX control. Please notice that some objects/identifier can't be browsed by the exPropertiesList control so, the Select method may fail. Also, if the exPropertiesList browser is empty when running the form, means that the control has not browse able properties, instead it may have methods or properties with parameters, that can't be browsed by the exPropertiesList control.

- Using the Exontrol's exPropertiesList control to get the list of interfaces that the inner object implements so it can guide you to the proper documentation. For instance, let's say that we need to find out something about a HTML fragment like: "MSHTML: <HTML><BODY>This is a line of text</BODY></HTML>", so you will need to run a code like follows:

```
Private Sub Form_Load()
```

```
    With Grid1
        .BeginUpdate
        .Columns.Add "Column 1"
        .Items.InsertControlItem , "MSHTML:<HTML><BODY>This is a line of
text</BODY></HTML>"
        .EndUpdate
        With .Items
            Debug.Print PropertiesList1.Interfaces(.ItemObject(.ItemByIndex(0)))
        End With
    End With
End Sub
```

Once that you run the sample, the output window (Immediate window) lists the interfaces that inner object implements, and so in our case we will notice a list like follows:

```
IUnknown
IPersistFile
IPersist
IViewObject
IDataObject
IOleObject
IOleInPlaceObject
IOleWindow
IOleInPlaceActiveObject
IParseDisplayName
IOleContainer
IOleItemContainer
IOleCache
IViewObject2
IOleCache2
IDispatch
IShellPropSheetExt
IOleInPlaceObjectWindowless
ICustomDoc
IHTMLDocument3
IFPMarkupServices
DispHTMLAnchorElement
```

DispHTMLAreaElement
DispHTMLBaseFontElement
DispHTMLBlockElement
DispHTMLBody
DispHTMLTableCaption
DispIHTMLOptionButtonElement
DispHTMLCommentElement
DispHTMLDDElement
DispHTMLDivElement
DispHTMLDTElement
DispHTMLDivPosition
DispHTMLFormElement
DispHTMLStyleElement
DispHTMLFontElement
DispHTMLFrameElement
DispHTMLFrameSetSite
DispHTMLHeaderElement
DispHTMLTitleElement
DispHTMLMetaElement
DispHTMLBaseElement
DispHTMLIsIndexElement
DispHTMLNextIdElement
DispHTMLIFrame
DispHTMLImg
DispIHTMLInputImage
DispIHTMLInputButtonElement
DispHTMLButtonElement
DispIHTMLInputTextElement
DispHTMLTextAreaElement
DispHTMLLabelElement
DispHTMLLIElement
DispHTMLLinkElement
DispHTMLListElement
DispHTMLMapElement
DispHTMLMarqueeElement
DispHTMLNoShowElement
DispHTMLObjectElement

DispHTMLOListElement
DispHTMLOptionElement
DispHTMLParaElement
DispHTMLPhraseElement
DispHTMLEmbed
DispHTMLScriptElement
DispHTMLSelectElement
DispHTMLTable
DispHTMLTableCol
DispHTMLTableSection
DispHTMLTableRow
DispHTMLTableCell
DispHTMLTextElement
DispHTMLUListElement
DispHTMLUnknownElement
DispHTMLBRElement
DispHTMLDListElement
DispHTMLBGsound
DispHTMLHRElement
DispIHTMLTextContainer
DispIHTMLControlElement
DispHTMLFrameBase
DispIHTMLInputFileElement
DispHTMLSpanFlow
DispHTMLFieldSetElement
DispHTMLLegendElement
DispHTMLSpanElement
DispHTMLRichtextElement
DispHTMLCurrentStyle
DispCEventObj
DispHTMLStyle
DispHTMLRuleStyle
DispHTMLWindow2
DispHTMLWindowProxy
DispHTMLDocument
DispHTMLHtmlElement
DispHTMLHeadElement

DispHTMLGenericElement
DispHTMLDOMAttribute
DispHTMLDOMTextNode
DispHTMLAreasCollection
DispHTMLElementCollection
DispHTMLAttributeCollection
IFPMarkupContainer
DispHTCDefaultDispatch
DispHTCEventBehavior
DispDOMChildrenCollection
DispHTMLAppBehavior
DispHTMLInputElement
DispHTCDescBehavior
DispHTCPropertyBehavior
DispHTMLDocumentFragment
DispHTCAttachBehavior
DispHTCMethodBehavior
DispHTMLPopup
DispHTMLRenderStyle
DispHTMLDefaults
DispHTMLStyleSheet
DispHTMLDOMImplementation
DispHTMLParamElement
DispHTMLScreen
IHTMLDocument4
IHTMLDocument5
IHTMLDocument2
IPerPropertyBrowsing
IViewObjectEx
IInternetHostSecurityManager
IPointerInactive
IHTMLDocument
IServiceProvider
ITargetContainer
IHlinkTarget
IPersistMoniker
DataSource

```
IPersistStreamInit
IPersistHistory
IMonikerProp
IProvideClassInfo2
IDispatchEx
IProvideMultipleClassInfo
IProvideClassInfo
IConnectionPointContainer
IOleControl
ISpecifyPropertyPages
IOleDocument
IOleDocumentView
IOleCommandTarget
IObjectIdentity
IObjectSafety
ISupportErrorInfo
```

The list of interfaces you got is quite long, but gives you useful information to start looking for the proper documentation that we need. In this list we need to locate the interfaces that are derived from the IDispatch interface, because in any COM oriented language you can call a property of an object only if it is derived from IDispatch interface. The question is how can we know which interface derives from IDispatch interface. In case you locate a IDispatchEx interface in your list, it means that it is possible to find multiple interfaces that derives from the IDispatch, else if only a IDispatch is located, one single interface can derive from the IDispatch. The list of interfaces differs from the objects to objects. For instance, if you are running the same code but using the "Exontrol.grid" identifier the list of interfaces looks like follows:

```
IUnknown
IPersistStorage
IPersist
IViewObject
IDataObject
IOleObject
IOleInPlaceObject
IOleWindow
IOleInPlaceActiveObject
IViewObject2
IDispatch
```

```
IOleInPlaceObjectWindowless
IViewObjectEx
IPersistStreamInit
IGrid
IProvideClassInfo2
IProvideClassInfo
IConnectionPointContainer
IOleControl
ISpecifyPropertyPages
IObjectSafety
IQuickActivate
ISupportErrorInfo
```

In our case the IHTMLDocument, IHTMLDocument2, IHTMLDocument3, IHTMLDocument4, IHTMLDocument5 interfaces derive from IDispatch so we can call any of properties of these interfaces like in the following code:

```
Private Sub Form_Load()
   With Grid1
      .BeginUpdate
      .Columns.Add "Column 1"
      .Items.InsertControlItem , "MSHTML:<HTML><BODY>This is a line of
text</BODY></HTML>"
      .EndUpdate
      With .Items
         Debug.Print .ItemObject(.ItemByIndex(0)).uniqueID()
      End With
   End With
End Sub
```

We arbitrary choose the uniqueID property that's a property of the IHTMLDocument3 interface. The documentation for these interfaces can be found in your MSDN documentation or by looking on the net for these names. Another alternative is using the OLEView tool in order to list the object's type library. **Unfortunately, You need to contact the vendor for particular ActiveX controls, because we can't provide documentation for any ActiveX control you might use. We can provide documentation only for our components.** Let's say that you will need other HTML fragment like: "MSHTML:<HTML><BODY bgcolor='#000000'>This is a line of text</BODY></HTML>", and you will ask us why the control can't change the HTML's

background color? Obviously, the control just passes the HTML fragment to the Web browser, and the Web browser handles in its own way. The idea is that the control is not responsible for the look and the behavior of the inner ActiveX controls. Anyway, in this particular case how can we solve the problem? If we see that some attributes are not recognized, we can try to look for a property that can do the same thing. In this case we are looking for the bkColor property of IHTMLDocument2 interface that sets or retrieves a value that indicates the background color behind the object. So the code should look like:

```
Private Sub Form_Load()
   With Grid1
      .BeginUpdate
      .Columns.Add "Column 1"
      .Items.InsertControlItem , "MSHTML:<HTML><BODY>This is a line of
text</BODY></HTML>"
      .EndUpdate
      With .Items
         With .ItemObject(.ItemByIndex(0))
            .bgColor = RGB(0, 0, 0)
            .fgcolor = RGB(255, 255, 255)
         End With
      End With
   End With
End Sub
```

Once an item of ActiveX type has been added you can get the OLE control that was created using the ItemObject property. Use the ItemControlID property to check if an item contains an ActiveX control. Use the ItemHeight property to change the item's height (the item's height manages the control's height). When the control contains an item of ActiveX type, it is recommend that you set the ScrollBySingleLine property of the 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 ExGrid control. The InsertControlItem method is not available if the control is running in the virtual mode.

The following VB sample adds dynamically an ExGrid ActiveX Control and a Microsoft Calendar Control:

```
' Inserts a new ActiveX control of Exontrol.Grid type
Dim hGrid As HITEM
```

```
hGrid = Grid1.Items.InsertControlItem(Grid1.Items(0), "Exontrol.Grid", runtimelicensekey)
' Sets the ActiveX control height
Grid1.Items.ItemHeight(hGrid) = 212
' Gets the ExGrid control created. Since the ProgID used to create the item is
"Exontrol.Grid"
' the object will be of EXGRIDLibCtl.Grid type
Dim objGrid As Object
Set objGrid = Grid1.Items.ItemObject(hGrid)
objGrid.Columns.Add "Column"
objGrid.Items.AddItem "One"
objGrid.Items.AddItem "Two"
objGrid.Items.AddItem "Three"

' Inserts a new ActiveX control of MSCAL.Calendar type
Dim hCalc As HITEM
hCalc = objGrid.Items.InsertControlItem(, "MSCal.Calendar")
Set objCalc = Grid1.Items.ItemObject(hCalc)
objCalc.ShowTitle = False
objCalc.ShowDateSelectors = False
```

where the runtimelicensekey is the exGrid's runtime license key. Please contact us to get the exGrid's runtime license key. **Your order number, or your registered e-mail address is required**, when requesting the control's runtime license key. Only, the vendor of the component you are going to use is able to give you such of runtime license, so please contact the control's vendor for such of key. Your development license key is not compatible with the runtime license key, so it can't be used here. Please notice that your development license key **is not equivalent** with the generated 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 shows how to handle any event that a contained ActiveX fires:

```
Private Sub Grid1_ItemOleEvent(ByVal Item As EXGRIDLibCtl.HITEM, ByVal Ev As
EXGRIDLibCtl.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 container window. For instance, the Web Brower added by the Grid1.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 = Grid1.Items.InsertControlItem(, "https://www.exontrol.com")
Grid1.Items.ItemHeight(hWeb) = 196
```

Next step is adding the AddItem event handler:

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
  If (Grid1.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
    Grid1.Items.ItemWindowHostCreateStyle(Item) =
Grid1.Items.ItemWindowHostCreateStyle(Item) + WS_HSCROLL + WS_VSCROLL
  End If
End Sub
```

The following VB sample adds the Exontrol's ExCalendar Component:

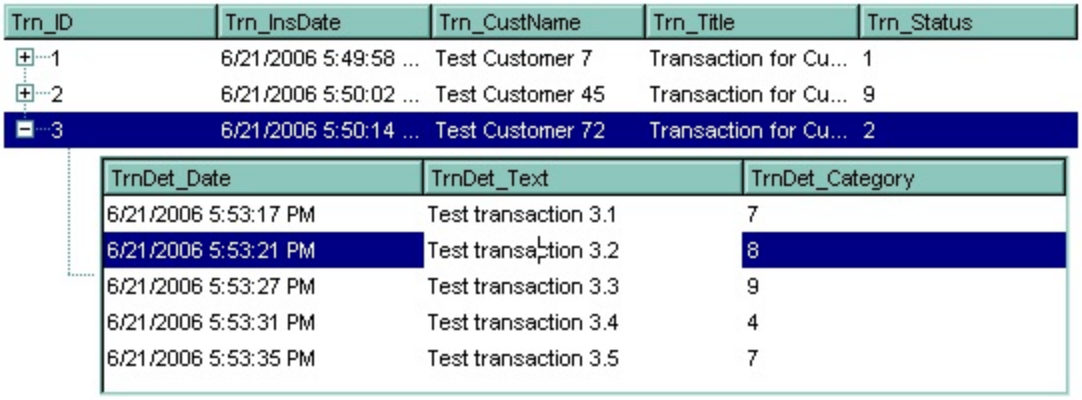
```
With Grid1
  .BeginUpdate
  .ScrollBySingleLine = True
  With Grid1.Items
    Dim h As HITEM
```

```
        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 VB sample binds the master control to a table, and displays related tables when the user expands an item/record. The sample uses the DataSource property to bind a record set to the control. The InsertControlItem method inserts an ActiveX inside the item.

```
Option Explicit

Public Function getRS(ByVal q As String) As Object
    Dim rs As Object, strDatabase
    strDatabase = App.Path + "\ExontrolDemo.mdb"
    Set rs = CreateObject("ADODB.Recordset")
    rs.Open q, "Provider = Microsoft.Jet.OLEDB.4.0; Data Source =" & strDatabase, 3, 3, 0
    Set getRS = rs
End Function

Private Sub Form_Load()
With Grid1
    .BeginUpdate
    .LinesAtRoot = exLinesAtRoot
    .MarkSearchColumn = False
    .ScrollBySingleLine = True
    .HideSelection = True
    Set .DataSource = getRS("Transactions")
    .EndUpdate
End With
End Sub
```

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
   With Grid1.Items
      .ItemHasChildren(Item) = .ItemParent(Item) = 0
   End With
End Sub

Private Sub Grid1_BeforeExpandItem(ByVal Item As EXGRIDLibCtl.HITEM, Cancel As
Variant)
   With Grid1.Items
      If .ItemHasChildren(Item) Then
         With .ItemObject(.InsertControlItem(Item, "Exontrol.Grid"))
            .BeginUpdate
               .MarkSearchColumn = False
               .HideSelection = True
               Set .DataSource = getRS("Select * from TransactionDetails where TrnDet_ID = "
& Grid1.Items.CellValue(Item, "Trn_ID"))
               .Columns(0).Visible = False
            .EndUpdate
         End With
         .ItemHasChildren(Item) = False
      End If
   End With
   Grid1.Refresh
End Sub
```



The following C++ sample adds the Exontrol's ExOrgChart Component:

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

```
#pragma warning( disable : 4146 )
#import <ExOrgChart.dll>

CItems items = m_grid.GetItems();
m_grid.BeginUpdate();
m_grid.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_grid.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 ExGrid Component:

```
axGrid1.BeginUpdate();
EXGRIDLib.Items items = axGrid1.Items;
axGrid1.ScrollBySingleLine = true;
int h = items.InsertControlItem(0, "Exontrol.Grid", "");
items.set_ItemHeight(h, 182);
object gridInside = items.get_ItemObject(h);
if (gridInside != null)
{
```

```
    EXGRIDLib.Grid grid = gridInside as EXGRIDLib.Grid;
    if (grid != null)
    {
        grid.BeginUpdate();
        grid.LinesAtRoot = EXGRIDLib.LinesAtRootEnum.exLinesAtRoot;
        grid.Columns.Add("Column 1");
        grid.Columns.Add("Column 2");
        grid.Columns.Add("Column 3");
        EXGRIDLib.Items itemsInside = grid.Items;
        int hInside = itemsInside.AddItem("Item 1");
        itemsInside.set_CellValue(hInside, 1, "SubItem 1");
        itemsInside.set_CellValue(hInside, 2, "SubItem 2");
        hInside = itemsInside.InsertItem(hInside, null, "Item 2");
        itemsInside.set_CellValue(hInside, 1, "SubItem 1");
        itemsInside.set_CellValue(hInside, 2, "SubItem 2");
        grid.EndUpdate();
    }
}
axGrid1.EndUpdate();
```

The following C# sample casts the ItemObject to IGrid interface:

```
int hX = axGrid1.Items.InsertControlItem(0, "Exontrol.Grid", "");
EXGRIDLib.IGrid spGrid = axGrid1.Items.get_ItemObject(hX) as EXGRIDLib.IGrid;
spGrid.Columns.Add("Inner Column");
```

The following VB.NET sample adds the Exontrol's ExOrgChart Component:

```
With AxGrid1
    .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 VB.NET sample casts the ItemObject to IGrid interface:

```
Dim hX As Long = .InsertControlItem(0, "Exontrol.Grid", "")
Dim spGrid As EXGRIDLib.IGrid = .ItemObject(hX)
spGrid.Columns.Add("Inner Column")
```

The following VFP sample adds the Exontrol's ExGrid Component:

```
with thisform.Grid1
  .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 VB6 sample shows you how to handle an event from a outer-inner-inner control. In other words, you have a master control (outer), which insert another control (inner), which insert another control (inner).

```
Private Sub expandItem(ByVal grid As Object, ByVal item As Long, ByVal level As Long)
  Debug.Print "Expand item in " & level & " control"
End Sub

' BeforeExpandItem event - Fired before an item is about to be expanded (collapsed).
Private Sub Grid1_BeforeExpandItem(ByVal item As EXGRIDLibCtl.HITEM, Cancel As
Variant)
  expandItem Grid1.Object, item, 0
End Sub

' ItemOleEvent event - Fired when an ActiveX control hosted by an item has fired an event.
Private Sub Grid1_ItemOleEvent(ByVal item As EXGRIDLibCtl.HITEM, ByVal Ev As
EXGRIDLibCtl.IOleEvent)
  With Grid1
    'Debug.Print Ev.ToString()
    If (Ev.ID = 12) Then   ' BeforeExpandItem
      expandItem Grid1.Items.ItemObject(item), Ev.Param(0).Value, 1
    Else
      If (Ev.ID = 14) Then   ' ItemOLEEvent
        'Debug.Print Ev.Param(1).Value.ToString()
        If (Ev.Param(1).Value.ID = 12) Then    ' BeforeExpandItem
          'Debug.Print "Expand item in inner-inner control"
          expandItem Grid1.Items.ItemObject(item).Items.ItemObject(Ev.Param(0).Value),
Ev.Param(1).Value.Param(0).Value, 2
        End If
      End If
    End If
  End With
End Sub
```

method Items.InsertItem ([Parent as HITEM], [UserData as Variant], [Value as Variant])

Inserts a new item, and returns a handle to the newly created item.

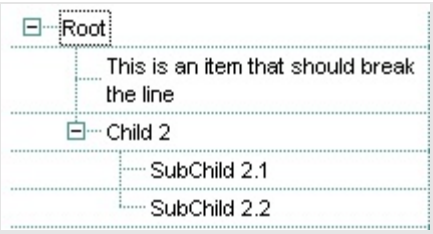
Type	Description
Parent as HITEM	A long expression that indicates the item's handle that indicates the parent item where the newly item is inserted
UserData as Variant	A Variant expression that indicates the item's extra data. Use the ItemData property to retrieve later this value.
Value as Variant	A Variant expression that indicates the cell's value on the first column, or a safe array that holds values for each column.

Return	Description
HITEM	Retrieves the handle of the newly created item.

Use the InsertItem property to add a new child to an 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. To insert an ActiveX control, use the InsertControlItem property of the Items property. Use the CellValue property to specify the values for cells in the second, third columns, and so on. Use the CellValueFormat property to specify whether the value contains HTML format or computed fields. The InsertItem method is not available if the control is running in the virtual mode. Use the LockedItemCount property to lock or unlock items to the top or bottom side of the control. Use the MergeCells method to combine one or more cells in a single cell. Use the SplitCell property to split a cell. If the CauseValidateValue property is True, the control fires the ValidateValue property when the user adds a new item. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

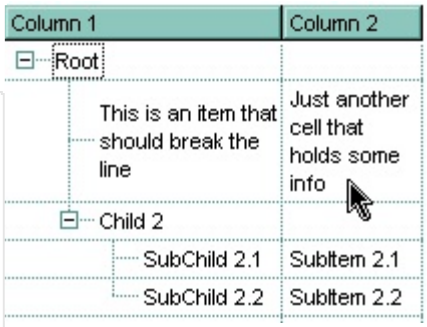
The following VB sample shows how to create a simple hierarchy (few items and one column):

```
With Grid1
  .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 Grid1
    .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 Grid1.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_grid.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 AxGrid1.Items
   Dim hItem As Integer = .InsertItem(.FocusItem, , "new child")
   .ExpandItem(.FocusItem) = True
End With
```

The following C# sample inserts a child item and expands the focused item:

```
int hItem = axGrid1.Items.InsertItem(axGrid1.Items.FocusItem, null, "new child");
axGrid1.Items.set_ExpandItem(axGrid1.Items.FocusItem, true);
```

The following VFP sample inserts a child item and expands the focused item:

```
with thisform.Grid1.Items
   .DefaultItem = .InsertItem( .FocusItem, "", "new child" )
   .DefaultItem = .FocusItem
   .ExpandItem(0) = .t.
endwith
```

method Items.InsertObjectItem (Parent as HITEM, [UserData as Variant], [Obj as Variant])

Inserts a new item that hosts the giving object, 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 object will be inserted. If the argument is missing then the InsertObjectItem property inserts the object as a root item. If the Parent property is referring a locked item (ItemLocked property), the InsertObjectItem property doesn't insert a new child, instead places the object to the locked item that's specified by the Parent property.
UserData as Variant	A VARIANT expression being specified at creating time, which can be accessed during the AddItem event. The ItemData property indicates the extra data associated with any item. The ItemData property is initalized with the value of the UserData parameter.
Obj as Variant	A object being hosted. The most common type is System.Windows.Forms.Control from the /NET framework. Generally, the Obj could be any control that can be placed to a form or dialog and it is visible at runtime. The Obj can not be a windowless control (a control that does not require a window, such a line or circle). The Obj parameter could be also an ActiveX control (that has already being placed in the form/dialog) in this case, the Obj should be the result of the property Object() (VB6, VFP). GetOcx() property. Finally, the Obj parameter could be of long type (numeric) in which case it should refer the handle of a window that follows to be hosted in the newly created item. The handle of the window can be obtained as m_hWnd member of MFC classes, hWnd or Handle property in the /NET framework. After creating the host, the ItemObject property can be used to retrieve the originally object (Obj parameter).

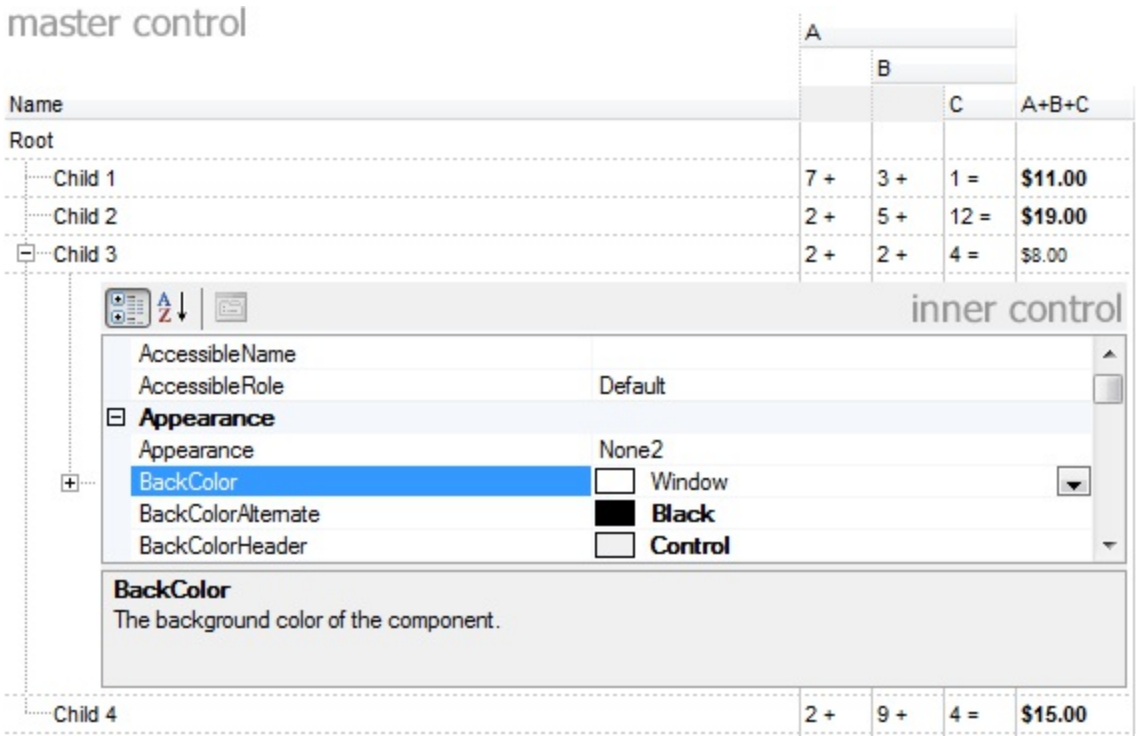
Return	Description
HITEM	A long expression that indicates the item's handle that indicates the newly created item.

The control supports /NET Control hosting, so you can insert any /NET component as a

child item of the control. This property is provided for the /NET assembly, but it is available for the /COM environment too. The InsertObjectItem property hosts the already created object to a new child item of the control while the InsertControlItem property creates the specified ActiveX control and hosts to a new child item of the control. So, the difference between the InsertObjectItem and InsertControlItem is that the InsertObjectItem does not create the object, while the InsertControlItem creates the specified control. If you are using the /NET assembly, the Obj should be the object to be inserted (usually of System.Windows.Forms.Control type), while for the /COM environment, the Obj should be the ActiveX control being already placed to a form, or a long expression that specifies the handle of the window to be hosted in a new child item of the control.

- The ItemHeight property specifes the height of the item, and so the height of the hosted object.
- The ItemWidth property specifies the width of hosted object, or the position/column in the item where the object is displayed.
- The ItemAllowSizing property indicates whether the user can resize the item at runtime, and so the object being hosted.
- The ItemObject property retrieves the originally object if the item was previously created using the InsertObjectItem property, or the created ActiveX control if using the InsertControlItem property.

The following screen shot shows the /NET assembly (master control) that hosts a System.Windows.Forms.PropertyGrid control from the /NET framework (inner control):



The following samples describes:

- inserting new child item to host your object
- handing the events for the inner controls

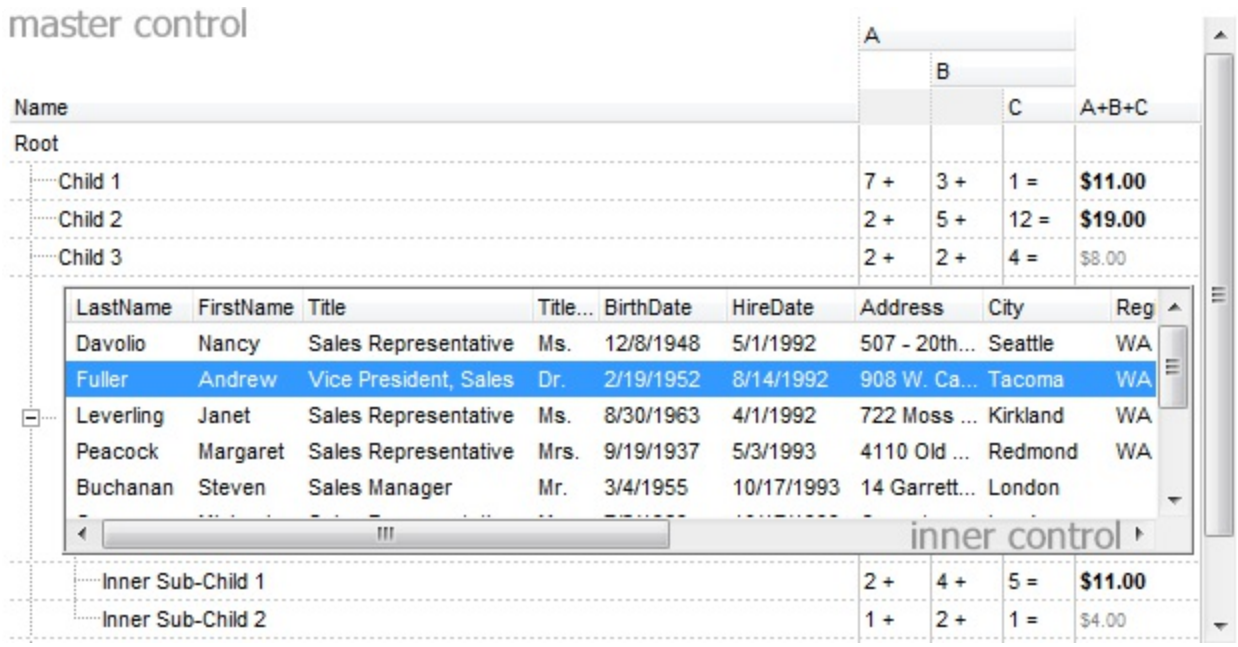
The following VB/NET sample inserts a child item that hosts an inner exgrid/net component:

```
With Exgrid1
   .BeginUpdate()
   With .Items
      Dim hx As Integer = .InsertObjectItem(.FocusItem, Nothing, New
exontrol.EXGRIDLib.exgrid())
      If (hx <> 0) Then
         With .get_ItemObject(hx)
            .BeginUpdate()
            .Columns.Add("inner column")
            .Items.AddItem("inner item")
            .EndUpdate()
         End With
      End If
   End With
   .EndUpdate()
End With
```

The following C# sample inserts a child item that hosts an inner exgrid/net component:

```
exgrid1.BeginUpdate();
int hx = exgrid1.Items.InsertObjectItem(exgrid1.Items.FocusItem, null, new
exontrol.EXGRIDLib.exgrid());
if ( hx != 0 )
{
   exontrol.EXGRIDLib.exgrid innerGrid = exgrid1.Items.get_ItemObject(hx) as
exontrol.EXGRIDLib.exgrid;
   if ( innerGrid != null )
   {
      innerGrid.BeginUpdate();
      innerGrid.Columns.Add("inner column");
      innerGrid.Items.AddItem("inner item");
      innerGrid.EndUpdate();
   }
}
exgrid1.EndUpdate();
```

The following screen shot shows the /NET assembly (master control) that hosts another /NET assembly control (inner control):



The following VB/NET sample inserts a child item that hosts an inner System.Windows.Forms.PropertyGrid component from the /NET framework:

```
With Exgrid1
   .BeginUpdate()
   With .Items
      Dim hx As Integer = .InsertObjectItem(.FocusItem, Nothing, New
System.Windows.Forms.PropertyGrid())
      If (hx <> 0) Then
         With .get_ItemObject(hx)
            .SelectedObject = Exgrid1
         End With
      End If
   End With
   .EndUpdate()
End With
```

The following C# sample inserts a child item that hosts an inner System.Windows.Forms.PropertyGrid component from the /NET framework:

```
exgrid1.BeginUpdate();
int hx = exgrid1.Items.InsertObjectItem(exgrid1.Items.FocusItem, null, new
System.Windows.Forms.PropertyGrid());
if ( hx != 0 )
```

```
{
   System.Windows.Forms.PropertyGrid innerPropertyGrid =
exgrid1.Items.get_ItemObject(hx) as System.Windows.Forms.PropertyGrid;
   if (innerPropertyGrid != null)
      innerPropertyGrid.SelectedObject = exgrid1;
}
exgrid1.EndUpdate();
```

You can handle the events for the inner controls as you would do if they has been placed to a form or dialog.

In C# you have to use the += operator while in VB/NET you can use the AddHandler as shown in the following samples:

The following VB/NET sample adds a handler for the DblClick event:

```
' innerGrid_DblClick handles event for the inner grid, where the sender is the object iself
AddHandler innerGrid.DblClick, AddressOf innerGrid_DblClick

Private Sub innerGrid_DblClick(ByVal sender As Object, ByVal Shift As Short, ByVal X As
Integer, ByVal Y As Integer)
   MessageBox.Show("innergrid dbl click")
End Sub
```

The following C# sample adds a handler for the DblClick event:

```
// innerGrid_DblClick handles event for the inner grid, where the sender is the object iself
innerGrid.DblClick += new
exontrol.EXGRIDLib.exgrid.DblClickEventHandler(innerGrid_DblClick);

void innerGrid_DblClick(object sender, short Shift, int X, int Y)
{
   MessageBox.Show("innergrid dbl click");
}
```

The sender parameter of the event identifies the object itself that fired the event, so you can use the handler for multiple instances of the same type. The sender will make the distinction. Now, in case you want to identify actually the item in the master control that hosts the sender, you can use the FindItemData property and set the UserData parameter of the InsertObjectItem property the same as for the Obj parameter. This way the

FindItemData(sender) will indicates the handle of the hosts the sender. In this case inserting the inner control should look like follows:

VB/NET

```
With Exgrid1
   .BeginUpdate()
   With .Items
      Dim innerGrid As exontrol.EXGRIDLib.exgrid = New exontrol.EXGRIDLib.exgrid()
      Dim hx As Integer = .InsertObjectItem(.FocusItem, innerGrid, innerGrid)
      If (hx <> 0) Then
         With innerGrid
            .BeginUpdate()
            .Columns.Add("inner column")
            .Items.AddItem("inner item")
            .EndUpdate()
         End With
      End If
      AddHandler innerGrid.DblClick, AddressOf innerGrid_DblClick
   End With
   .EndUpdate()
End With
```

C#

```
exgrid1.BeginUpdate();
exontrol.EXGRIDLib.exgrid innerGrid = new exontrol.EXGRIDLib.exgrid();
int hx = exgrid1.Items.InsertObjectItem(exgrid1.Items.FocusItem, innerGrid, innerGrid);
if (hx != 0)
{
   innerGrid.BeginUpdate();
   innerGrid.Columns.Add("inner column");
   innerGrid.Items.AddItem("inner item");
   innerGrid.EndUpdate();
}
innerGrid.DblClick += new
exontrol.EXGRIDLib.exgrid.DblClickEventHandler(innerGrid_DblClick);
```

```
exgrid1.EndUpdate();
```

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 Grid1_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 EXGRIDLibCtl.HitTestInfoEnum
  ' Gets the item from (X,Y)
  With Grid1
    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 OnMouseMoveGrid1(short Button, short Shift, long X, long Y)
{
   long c = 0, hit = 0, hItem = m_grid.GetItemFromPoint( X, Y, &c, &hit );
   if ( hItem != 0 )
   {
      CItems items = m_grid.GetItems();
      if ( items.GetIsItemLocked( hItem ) )
      {
         COleVariant vtItem( hItem ), vtColumn( c );
         CString strCaption = V2S( &items.GetCellValue( 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 AxGrid1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_MouseMoveEvent) Handles AxGrid1.MouseMoveEvent
   With AxGrid1
      Dim i As Integer, c As Integer, hit As EXGRIDLib.HitTestInfoEnum
      i = .get_ItemFromPoint(e.x, e.y, c, hit)
      If Not (i = 0) Then
         With .Items
            If (.IsItemLocked(i)) Then
               Dim strCaption As String = .CellCaption(i, c)
               Debug.WriteLine("Cell: " & strCaption & " 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 axGrid1_MouseMoveEvent(object sender,
```

```
AxEXGRIDLib._IGridEvents_MouseMoveEvent e)
{
   int c = 0;
   EXGRIDLib.HitTestInfoEnum hit;
   int i = axGrid1.get_ItemFromPoint(e.x, e.y, out c, out hit);
   if (i != 0)
      if (axGrid1.Items.get_IsItemLocked(i))
      {
         object cap = axGrid1.Items.get_CellValue(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 (add the code the MouseMove event):

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

local c, hit
c = 0
hit = 0
with thisform.Grid1
   .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, [Partially as Variant]) as Boolean

Checks if the specific item fits the control's client area.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Partially as Variant	A boolean expression that indicates whether the item is partially visible or not. If the Partially parameter is missing, the True value is used.
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:

```
Dim h As HITEM
Dim i As Long, j As Long, nCols As Long
nCols = Grid1.Columns.Count
With Grid1.Items
   h = .FirstVisibleItem
   While Not (h = 0) And .IsItemVisible(h)
      Dim s As String
      s = ""
      For j = 0 To nCols - 1
         s = s + .CellValue(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_grid.GetItems();
long hItem = items.GetFirstVisibleItem();
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
while ( hItem && items.GetIsItemVisible( hItem, vtMissing ) )
{
   OutputDebugString( V2S( &items.GetCellValue( COleVariant( hItem ), COleVariant(
long(0) ) ) ) );
   hItem = items.GetNextVisibleItem( hItem );
}
```

where the V2S function converts a VARIANT value to a string expression and looks like:

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
   if ( pv )
   {
      if ( pv->vt == VT_ERROR )
         return szDefault;

      COleVariant vt;
      vt.ChangeType( VT_BSTR, pv );
      return V_BSTR( &vt );
   }
   return szDefault;
}
```

The following VB.NET sample enumerates the items that fit the control's client area:

```
With AxGrid1.Items
   Dim hItem As Integer = .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:

```
EXGRIDLib.Items items = axGrid1.Items;
int hItem = items.FirstVisibleItem;
while ((hItem != 0) && (items.get_IsItemVisible(hItem, null)))
{
   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.Grid1.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 while it's of ActiveX type.

Type	Description
Item as HITEM	A long expression that indicates the item's handle that was previously created by InsertControlItem property.
AppearanceEnum	An AppearanceEnum value 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 item's handle.
Color	A color expression that indicates the item's background color.

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. 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 item's appearance. The sample uses the "".

```
With Grid1
   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_grid.GetVisualAppearance().Add( 0x50,
COleVariant(_T("D:\\Temp\\ExGrid.Help\\item.ebn")) );
m_grid.GetItems().SetItemBackColor( m_grid.GetItems().GetFirstVisibleItem() , 0x50000000
```

```
);
```

The following VB.NET sample changes the item's appearance:

```
With AxGrid1
   With .VisualAppearance
      .Add(&H50, "D:\Temp\ExGrid.Help\item.ebn")
   End With
   With .Items
      .ItemBackColor(.FirstVisibleItem) = &H50000000
   End With
End With
```

The following C# sample changes the item's appearance:

```
axGrid1.VisualAppearance.Add(0x50, "D:\\Temp\\ExGrid.Help\\item.ebn");
axGrid1.Items.set_ItemBackColor(axGrid1.Items.FirstVisibleItem, 0x50000000);
```

The following VFP sample changes the item's appearance:

```
With thisform.Grid1
   With .VisualAppearance
      .Add(80, "D:\Temp\ExGrid.Help\item.ebn")
   EndWith
   with .Items
      .DefaultItem = .FirstVisibleItem
      .ItemBackColor(0) = 1342177280
   endwith
EndWith
```

where the 1342177280 value represents the 0x50000000 hexa value.

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

You can use the following VB.NET function:

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

```
    i = i + 256 * 256 * c.B
    ToUInt32 = Convert.ToUInt32(i)
    End Function
```

You can use the following C# function:

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

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

```
axGrid1.Items.set_ItemBackColor(axGrid1.Items.FocusItem, ToUInt32(Color.Red) );
```

The following VB.NET sample changes the background color for the focused item:

```
With AxGrid1.Items
    .ItemBackColor(.FocusItem) = ToUInt32(Color.Red)
End With
```

The following VB sample changes the background color for the focused item:

```
With Grid1.Items
    .ItemBackColor(.FocusItem) = vbRed
End With
```

The following C++ sample changes the background color for the focused item:

```
#include "Items.h"
CItems items = m_grid.GetItems();
items.SetItemBackColor( items.GetFocusItem(), RGB(255,0,0) );
```

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

```
with thisform.Grid1.Items
```

```
    .DefaultItem = .FocusItem
    .ItemBackColor( 0 ) = RGB(255,0,0)
endwith
```

Use the following VB sample to change the background color for the first column when adding new items:

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
    Grid1.Items.CellBackColor(Item, o) = vbBlue
End Sub
```

property Items.ItemBold(Item as HITEM) as Boolean

Retrieves or sets a value that indicates whether the item is bolded.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Boolean	A boolean expression that indicates whether the item is bolded.

Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the CellValueFormat 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 Grid1_SelectionChanged()
   If Not (hOldBold = 0) Then
      Grid1.Items.ItemBold(hOldBold) = False
   End If
   hOldBold = Grid1.Items.SelectedItem()
   Grid1.Items.ItemBold(hOldBold) = True
End Sub
```

The following VB sample bolds the focused item:

```
With Grid1.Items
   .ItemBold(.FocusItem) = True
End With
```

The following C++ sample bolds the focused item:

```
#include "Items.h"
CItems items = m_grid.GetItems();
items.SetItemBold( items.GetFocusItem() , TRUE );
```

The following C# sample bolds the focused item:

```
axGrid1.Items.set_ItemBold(axGrid1.Items.FocusItem, true);
```

The following VB.NET sample bolds the focused item:

```
With AxGrid1.Items
   .ItemBold(.FocusItem) = True
End With
```

The following VFP sample bolds the focused item:

```
with thisform.Grid1.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 the Items collection..

Type	Description
Index as Long	A long value that indicates the item's index.
HITEM	A long expression that indicates the item's handle

Use the ItemByIndex to get the index of an item. Use the ItemCount property to count the items in the control. 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: Grid1.Items(0), Grid1.Items.ItemByIndex(0).

The following VB sample enumerates all items in the control:

```
Dim i As Long, n As Long
With Grid1.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_grid.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 AxGrid1
```

```
    Dim i As Integer
    For i = 0 To .Items.ItemCount - 1
        Debug.Print(.Items.CellValue(.Items(i), 0))
    Next
End With
```

The following C# sample enumerates all items in the control:

```
EXGRIDLib.Items items = axGrid1.Items;
for (int i = 0; i < items.ItemCount; i++)
{
    object caption = items.get_CellValue(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.Grid1.Items
    local i
    for i = 0 to .ItemCount - 1
        .DefaultItem = .ItemByIndex( i )
        wait window nowait .CellCaption(0,0)
    next
endwith
```

property Items.ItemCell (Item as HITEM, ColIndex as Variant) as HCELL

Retrieves the cell's handle given the item and the column.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
ColIndex as Variant	A long expression that indicates the the column's index, a string expression that indicates the column's caption or the column's key.
HCELL	A long value that indicates the cell's handle.

The ItemCell property retrieves the handle of the cell that belongs to the item on the specified column. The InnerCell properties always returns the handle to the master cells (master cell is a cell where the splitting starts). Use the SplitCell property to split a cell into multiple cells. Use the MergeCells property to merge multiple cells.

Note: The intersection of an item with a column defines a cell. Each cell is uniquely represented by its handle. The cell's handle is of HCELL type, that's equivalent with a long type. All properties of Items object that have two parameters *Item* and *ColIndex,* refer a cell.

The following lines are equivalents and each of them changes the bold font attribute of the first cell on the first item.

```
With Grid1
   .Items.CellBold(, .Items.ItemCell(.Items(0), 0)) = True
   .Items.CellBold(.Items(0), 0) = True
   .Items.CellBold(.Items(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0)) = True
   .Items.CellBold(.Items.ItemByIndex(0), 0) = True
   .Items.CellBold(.Items(0), Grid1.Columns(0).Caption) = True
End With
```

property Items.ItemChild (Item as HITEM) as HITEM

Retrieves the first child item of a specified item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
HITEM	A long expression that indicates the item's handle that indicates the first child item of the Item

If the ItemChild property gets 0, the item has no child items. Use the ItemChild property to get the first child of an item. The NextVisibleItem or NextSiblingItem gets the next visible, sibling item. Use the ChildCount property to count the number of child items. Use the ItemHasChildren property to built a virtual grid. A virtual grid loads items when the user expands an item. Use the ItemParent property to retrieve the handle of the parent item. The control displays a +/- sign to parent items, if the HasButtons property is not zero, the ItemChild property is not empty, or the ItemHasChildren property is True.

The following VB function recursively enumerates the item and all its child items:

```
Sub RecItem(ByVal c As EXGRIDLibCtl.Grid, 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( CGrid* pGrid, long hItem )
{
   COleVariant vtColumn( (long)0 );
   if ( hItem )
   {
```

```
        CItems items = pGrid->GetItems();

        CString strCaption = V2S( &items.GetCellValue( COleVariant( hItem ), vtColumn ) ),
strOutput;
        strOutput.Format( "Cell: '%s'\n", strCaption );
        OutputDebugString( strOutput );

        long hChild = items.GetItemChild( hItem );
        while ( hChild )
        {
            RecItem( pGrid, 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 AxEXGRIDLib.AxGrid, 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(AxEXGRIDLib.AxGrid  grid, int hItem)
{
    if (hItem != 0)
    {
```

```
        EXGRIDLib.Items items = grid.Items;
        object caption = items.get_CellValue( hItem, 0 );
        System.Diagnostics.Debug.WriteLine(caption != null ? caption.ToString() : "");

        int hChild = items.get_ItemChild(hItem);
        while (hChild != 0)
        {
            RecItem(grid, 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.Grid1
    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.
String	A string expression that indicates the control identifier used by InsertControlItem property 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 Items collection

The ItemCount property counts the items in the control. Use the ItemByIndex property to access an item giving its index. Use the AddItem, InsertItem, InsertControlItem, PutItems or DataSource property to add new items to the control. Use ChildCount to get the number of child items.

The following VB sample enumerates all items in the control:

```
Dim i As Long, n As Long
With Grid1.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_grid.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 AxGrid1
   Dim i As Integer
```

```
    For i = 0 To .Items.ItemCount - 1
        Debug.Print(.Items.CellValue(.Items(i), 0))
    Next
End With
```

The following C# sample enumerates all items in the control:

```
EXGRIDLib.Items items = axGrid1.Items;
for (int i = 0; i < items.ItemCount; i++)
{
    object caption = items.get_CellValue(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.Grid1.Items
    local i
    for i = 0 to .ItemCount - 1
        .DefaultItem = .ItemByIndex( i )
        wait window nowait .CellCaption(0,0)
    next
endwith
```

property Items.ItemData(Item as HITEM) as Variant

Retrieves or sets the extra data for a specific item.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item.
Variant	A variant value that indicates the item's extra data.

Use the ItemData property to assign an extra value to an item. Use CellData property to associate an extra data with a cell. The ItemData and CellData are of Variant type, so you will be able to save here what ever you want: numbers, objects, strings, and so on. The user data is only for user use. The control doesn't use this value. Use the Data property to assign an extra data to a column. For instance, you can use the RemoveItem event to release any extra data that is associated to the item.

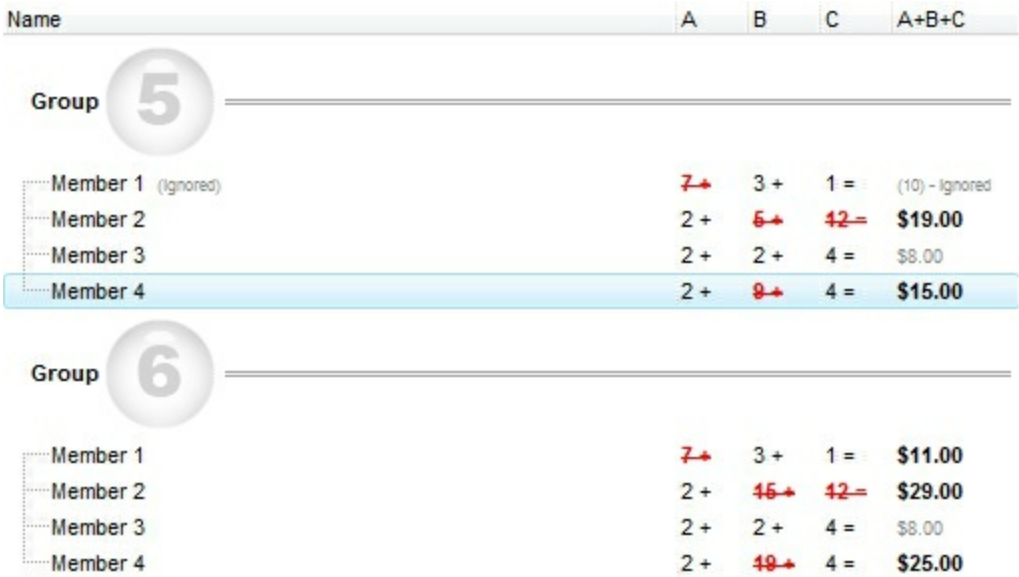
property Items.ItemDivider(Item as HITEM) as Long

Specifies whether the item acts like a divider or normal item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Long	A long expression that indicates the column's index.

A divider item uses the item's client area to display a single cell. You can use the ItemDivider property to separate the items, display groups of items or display total or subtotals fields. The ItemDivider property specifies the index of the cell being displayed in the item's client area. In other words, the divider item merges the item cells into a single cell. The CellHAlignment property specifies the horizontal alignment for the cell's content. 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 SortableItem property specifies whether the item keeps its position after sorting.

The following screen shot shows the items arranged as groups (Group 5 and Group 6 are divider items):



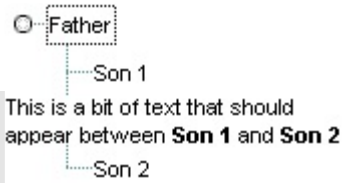
The following screen shot shows subtotals items (42 and 73 are divider items):

Name	A	B	C	A+B+C
Group 1				
Member 1 (Ignored)	~~7 +~~	3 +	1 =	(10) - Ignored
Member 2	2 +	~~5 +~~	~~12 =~~	**$19.00**
Member 3	2 +	2 +	4 =	$8.00
Member 4	2 +	~~9 +~~	4 =	**$15.00**
				$42.00
Group 2				
Member 1	~~7 +~~	3 +	1 =	**$11.00**
Member 2	2 +	~~15 +~~	~~12 =~~	**$29.00**
Member 3	2 +	2 +	4 =	$8.00
Member 4	2 +	~~19 +~~	4 =	**$25.00**
				$73.00

The following screen shot shows subtotals items in combination with EBN features:

Name	A	B	C	A+B+C
Group 1				
Member 1 (Ignored)	~~7 +~~	3 +	1 =	(10) - Ignored
Member 2	2 +	~~5 +~~	~~12 =~~	**$19.00**
Member 3	2 +	2 +	4 =	$8.00
Member 4	2 +	~~9 +~~	4 =	**$15.00**
				$42.00
Group 2				
Member 1	~~7 +~~	3 +	1 =	**$11.00**
Member 2	2 +	~~15 +~~	~~12 =~~	**$29.00**
Member 3	2 +	2 +	4 =	$8.00
Member 4	2 +	~~19 +~~	4 =	**$25.00**
				$73.00

The following VB sample adds a divider item between two child items:




```
Private Sub Form_Load()
  With Grid1
    .BeginUpdate
      .DefaultItemHeight = 24
      .MarkSearchColumn = False
      .HeaderVisible = False
      .LinesAtRoot = exLinesAtRoot
      .HasButtons = exCircle
      With .Columns
        With .Add("Column 1")
```

```
            .SortType = EXGRIDLibCtl.SortTypeEnum.SortCellData
        End With
        With .Add("Column 2")
            .Visible = False
        End With
    End With

    With .Items
        Dim h As HITEM, g As HITEM
        h = .InsertItem(, , "Father")
        .InsertItem h, 1, "Son 1"
        g = .InsertItem(h, 50, "")
        .CellValue(g, 1) = "This is a bit of text that should appear
between <b>Son 1</b> and <b>Son 2</b>"
        .CellValueFormat(g, 1) = exHTML
        .CellSingleLine(g, 1) = False
        .ItemDivider(g) = 1
        .ItemDividerLine(g) = EmptyLine
        .InsertItem h, 99, "Son 2"
        .ExpandItem(h) = True
    End With
  .EndUpdate
End With
End Sub
```

The following VB sample adds a divider item, that's not selectable too:

```
With Grid1.Items
  Dim i As Long
  i = .AddItem("divider item")
  .ItemDivider(i) = 0
  .SelectableItem(i) = False
End With
```

The following C++ sample adds a divider item, that's not selectable too:

```
#include "Items.h"
CItems items = m_grid.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 = axGrid1.Items.AddItem("divider item");
axGrid1.Items.set_ItemDivider(i, 0);
axGrid1.Items.set_SelectableItem(i, false);
```

The following VB.NET sample adds a divider item, that's not selectable too:

```
With AxGrid1.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.Grid1.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.

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.ItemFiltered (Item as HITEM) as Boolean

Checks whether the item is included in the control's filter.

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 filtered.

Use the ItemFiltered property to check whether an item is included in the control's filter. Use the FilterType property to specify the type of filter that's applied to a column. The ApplyFilter method should be called to update the control's content after changing the Filter or FilterType property. The ItemCount property counts the items in the control's list. Use the ItemByIndex property to access an item giving its index.

The following VB sample enumerates all items that are not included in the list when a filter is applied:

```
Dim i As Long
With Grid1.Items
   For i = 0 To .ItemCount - 1
      Dim h As EXGRIDLibCtl.HITEM
      h = .ItemByIndex(i)
      If (Not .ItemFiltered(h)) Then
         Debug.Print .CellValue(h, 0)
      End If
   Next
End With
```

The following C++ sample enumerates all items that are not included in the list when a filter is applied:

```
#include "Items.h"
CItems items = m_grid.GetItems();
for ( long i = 0; i < items.GetItemCount(); i++ )
{
   long hItem = items.GetItemByIndex( i );
   if ( !items.GetItemFiltered( hItem ) )
   {
      COleVariant vtItem( hItem ), vtColumn( long(0) );
```

```
        CString strFormat;
        strFormat.Format( "'%s' is not included\r\n", V2S( &items.GetCellValue( vtItem,
vtColumn ) ) );
        OutputDebugString( strFormat );
    }
}
```

The following VB.NET sample enumerates all items that are not included in the list when a filter is applied:

```
Private Sub AxGrid1_ClickEvent(ByVal sender As Object, ByVal e As System.EventArgs)
Handles AxGrid1.ClickEvent
    Dim i As Long
    With AxGrid1.Items
        For i = 0 To .ItemCount - 1
            Dim h As Integer = .ItemByIndex(i)
            If (Not .ItemFiltered(h)) Then
                Dim cellValue As Object = .CellValue(h, 0)
                Dim strValue As String = ""
                If Not (cellValue Is Nothing) Then
                    strValue = cellValue.ToString()
                End If
                Debug.WriteLine(strValue)
            End If
        Next
    End With
End Sub
```

The following C# sample enumerates all items that are not included in the list when a filter is applied:

```
EXGRIDLib.Items items = axGrid1.Items;
for (int i = 0; i < items.ItemCount; i++)
    if (!items.get_ItemFiltered(items[i]))
    {
        object cellValue = axGrid1.Items.get_CellValue(i, 0);
        string strValue = cellValue != null ? cellValue.ToString() : "";
        System.Diagnostics.Debug.WriteLine(strValue);
```

```
}
```

The following VFP sample enumerates all items that are not included in the list when a filter is applied:

```
with thisform.Grid1.Items
   local i
   For i = 0 To .ItemCount - 1
      local h
      h = .ItemByIndex(i)
      If (!.ItemFiltered(h)) Then
         wait window nowait .CellValue(h, 0)
      EndIf
   Next
endwith
```

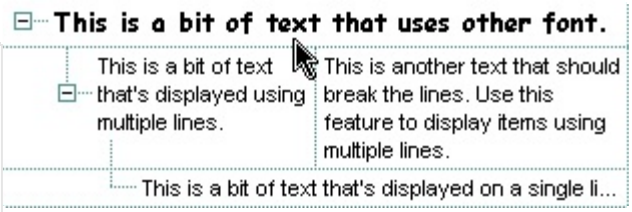
property Items.ItemFont (Item as HITEM) as IFontDisp

Retrieves or sets the item's font.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
IFontDisp	A font object being used.

By default, the ItemFont property is nothing. If the ItemFont property is nothing, the item uses the control's font. Use the ItemFont property to define a different font for the item. Use the CellFont and ItemFont properties to specify different fonts for cells or items. Use the CellBold, CellItalic, CellUnderline, CellStrikeout, ItemBold, ItemUnderline, ItemStrikeout, ItemItalic or CellValueFormat 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 ItemHeight property to specify the height of the item.

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




```
With Grid1.Items
   .ItemFont(.FocusItem) = Grid1.Font
   With .ItemFont(.FocusItem)
      .Name = "Comic Sans MS"
      .Bold = True
   End With
End With
Grid1.Refresh
```

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

```
#include "Items.h"
#include "Font.h"
CItems items = m_grid.GetItems();
items.SetItemFont( items.GetFocusItem(), m_grid.GetFont().m_lpDispatch );
COleFont font = items.GetItemFont( items.GetFocusItem() );
font.SetName( "Comic Sans MS" );
font.SetBold( TRUE );
m_grid.Refresh();
```

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

```
With AxGrid1.Items
   .ItemFont(.FocusItem) = IFDH.GetIFontDisp(AxGrid1.Font)
   With .ItemFont(.FocusItem)
      .Name = "Comic Sans MS"
      .Bold = True
   End With
End With
AxGrid1.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:

```
axGrid1.Items.set_ItemFont( axGrid1.Items.FocusItem, IFDH.GetIFontDisp( axGrid1.Font ) );
stdole.IFontDisp spFont = axGrid1.Items.get_ItemFont(axGrid1.Items.FocusItem );
spFont.Name = "Comic Sans MS";
spFont.Bold = true;
axGrid1.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.Grid1.Items
    .DefaultItem = .FocusItem
    .ItemFont(0) = thisform.Grid1.Font
    with .ItemFont(0)
        .Name = "Comic Sans MS"
        .Bold = .t.
    endwith
endwith
thisform.Grid1.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. Use the ConditionalFormats method to apply formats to a cell or range of cells, and have that formatting change depending on the value of the cell or the value of a formula.

The following VB sample changes the foreground color for cells in the first column as user add new items:

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
  Grid1.Items.CellForeColor(Item, o) = 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:

```
axGrid1.Items.set_ItemForeColor(axGrid1.Items.FocusItem, ToUInt32(Color.Red) );
```

The following VB.NET sample changes the foreground color of the focused item:

```
With AxGrid1.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_grid.GetItems();
items.SetItemForeColor( items.GetFocusItem(), RGB(255,0,0) );
```

The following VFP sample changes the foreground color of the focused item:

```
with thisform.Grid1.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 grid. 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 the user expands an item (the sample has effect only if your control contains items that have the ItemHasChildren property on True):

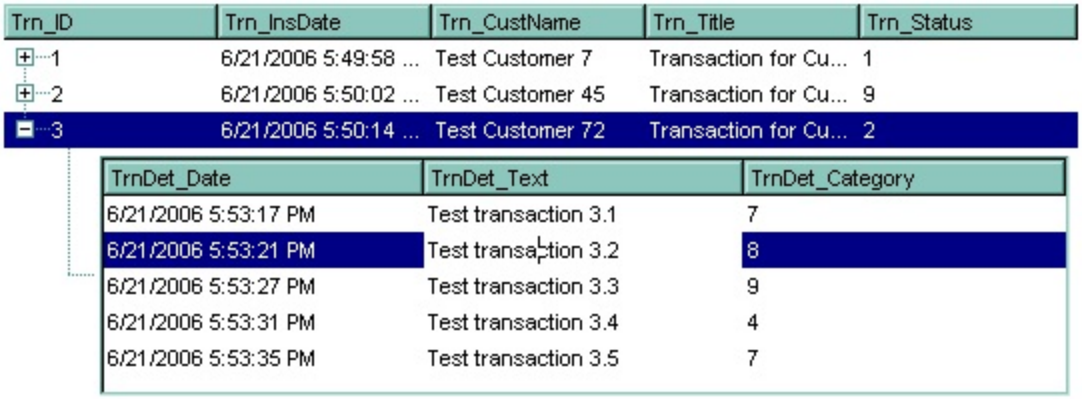
```
Private Sub Grid1_BeforeExpandItem(ByVal Item As EXGRIDLibCtl.HITEM, Cancel As
Variant)
  With Grid1.Items
    If (.ItemHasChildren(Item)) Then
      If .ChildCount(Item) = 0 Then
        Dim h As Long
        h = .InsertItem(Item, , "new " & Item)
      End If
    End If
  End With
End Sub
```

The following VB sample binds the master control to a table, and displays related tables when the user expands an item/record. The sample uses the DataSource property to bind a record set to the control. The InsertControlItem method inserts an ActiveX inside the item.

```
Option Explicit
```

```
Public Function getRS(ByVal q As String) As Object
   Dim rs As Object, strDatabase
   strDatabase = App.Path + "\ExontrolDemo.mdb"
   Set rs = CreateObject("ADODB.Recordset")
   rs.Open q, "Provider = Microsoft.Jet.OLEDB.4.0; Data Source =" & strDatabase, 3, 3, 0
   Set getRS = rs
End Function

Private Sub Form_Load()
With Grid1
   .BeginUpdate
   .LinesAtRoot = exLinesAtRoot
   .MarkSearchColumn = False
   .ScrollBySingleLine = True
   .HideSelection = True
   Set .DataSource = getRS("Transactions")
   .EndUpdate
End With
End Sub

Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
   With Grid1.Items
      .ItemHasChildren(Item) = .ItemParent(Item) = 0
   End With
End Sub

Private Sub Grid1_BeforeExpandItem(ByVal Item As EXGRIDLibCtl.HITEM, Cancel As
Variant)
   With Grid1.Items
      If .ItemHasChildren(Item) Then
         With .ItemObject(.InsertControlItem(Item, "Exontrol.Grid"))
            .BeginUpdate
               .MarkSearchColumn = False
               .HideSelection = True
               Set .DataSource = getRS("Select * from TransactionDetails where TrnDet_ID = "
& Grid1.Items.CellValue(Item, "Trn_ID"))
               .Columns(0).Visible = False
```

```
            .EndUpdate
        End With
        .ItemHasChildren(Item) = False
    End If
End With
Grid1.Refresh
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 AxGrid1_BeforeExpandItem(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_BeforeExpandItemEvent) Handles AxGrid1.BeforeExpandItem
    With AxGrid1.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 axGrid1_BeforeExpandItem(object sender,
AxEXGRIDLib._IGridEvents_BeforeExpandItemEvent e)
{
    EXGRIDLib.Items items = axGrid1.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 OnBeforeExpandItemGrid1(long Item, VARIANT FAR* Cancel)
{
    CItems items = m_grid.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.Grid1.Items
    if ( .ItemHasChildren( item ) )
        if ( .ChildCount( item ) = 0 )
            .InsertItem(item,"","new " + trim(str(item)))
        endif
    endif
endwith
```

property Items.ItemHeight(Item as HITEM) as Long

Retrieves or sets the item's height.

Type	Description
Item as HITEM	A long expression that indicates the handle of the item.
Long	A long value that indicates the item's height.

To change the default height of the item before inserting it into the items collection you can call DefaultItemHeight property. The control supports items with different heights. When an item hosts an ActiveX control (was previously created by the InsertControlItem property), the ItemHeight property changes the height of contained ActiveX control too. *If the CellSingleLine property is False, the ItemHeight property has* ***no*** *effect. The Column.Def(exCellPaddingTop) and Column.Def(exCellPaddingBottom) defines the vertical padding.* Use the ItemMaxHeight property to specify the maximum height of the item when its height is variable. Use the CellVAlignment property to align vertically a cell. Use the ScrollBySingleLine property when using items with different heights. Use the SelectableItem property to specify whether the user can select an item. For instance, in order to hide an item you can set the ItemHeight property on 0, and SelectableItem property on False. Use the ItemAllowSizing property to specify whether the user can resize the item at runtime.

property Items.ItemItalic(Item as HITEM) as Boolean

Retrieves or sets a value that indicates whether the item should appear in italic.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Boolean	A boolean expression that indicates whether the item's font attributes include Italic attribute.

Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the CellValueFormat 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 Grid1_SelectionChanged()
   If Not (h = 0) Then Grid1.Items.ItemItalic(h) = False
   h = Grid1.Items.SelectedItem()
   Grid1.Items.ItemItalic(h) = True
End Sub
```

The following VB sample makes italic the focused item:

```
With Grid1.Items
   .ItemItalic(.FocusItem) = True
End With
```

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

```
#include "Items.h"
CItems items = m_grid.GetItems();
items.SetItemItalic( items.GetFocusItem() , TRUE );
```

The following C# sample makes italic the focused item:

```
axGrid1.Items.set_ItemItalic(axGrid1.Items.FocusItem, true);
```

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

```
With AxGrid1.Items
   .ItemItalic(.FocusItem) = True
End With
```

The following VFP sample makes italic the focused item:

```
with thisform.Grid1.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 item contains cells with CellSingleLine property on False. Use the ItemHeight property to get the item's height. Use the CellVAlignment property to align vertically a cell. Use the DefaultItemHeight property to specify the default height for all items before loading items.

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 Variant

Retrieves the item's ActiveX object, if the item was previously created by InsertControlItem property, or the original object being used when calling the InsertObjectItem property.

Type	Description
Item as HITEM	A long expression that indicates the item's handle that was previously created by InsertControlItem or InsertObjectItem property.
Variant	An object that represents the ActiveX hosted by the item

The ItemObject retrieves the ActiveX object being created by the InsertControlItem method, or the object being hosted when using the InsertObjectItem property. 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 Grid1
  .BeginUpdate
  .ScrollBySingleLine = True
  With Grid1.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_grid.GetItems();
m_grid.BeginUpdate();
m_grid.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_grid.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 ExGrid Component:

```
axGrid1.BeginUpdate();
EXGRIDLib.Items items = axGrid1.Items;
axGrid1.ScrollBySingleLine = true;
int h = items.InsertControlItem(0, "Exontrol.Grid", "");
items.set_ItemHeight(h, 182);
object gridInside = items.get_ItemObject(h);
if (gridInside != null)
{
```

```
    EXGRIDLib.Grid grid = gridInside as EXGRIDLib.Grid;
    if (grid != null)
    {
        grid.BeginUpdate();
        grid.LinesAtRoot = EXGRIDLib.LinesAtRootEnum.exLinesAtRoot;
        grid.Columns.Add("Column 1");
        grid.Columns.Add("Column 2");
        grid.Columns.Add("Column 3");
        EXGRIDLib.Items itemsInside = grid.Items;
        int hInside = itemsInside.AddItem("Item 1");
        itemsInside.set_CellValue(hInside, 1, "SubItem 1");
        itemsInside.set_CellValue(hInside, 2, "SubItem 2");
        hInside = itemsInside.InsertItem(hInside, null, "Item 2");
        itemsInside.set_CellValue(hInside, 1, "SubItem 1");
        itemsInside.set_CellValue(hInside, 2, "SubItem 2");
        grid.EndUpdate();
    }
}
axGrid1.EndUpdate();
```

The following VB.NET sample adds the Exontrol's ExOrgChart Component:

```
With AxGrid1
    .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.Grid1
   .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 item's parent item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
HITEM	A long expression that indicates the item's handle that indicates the parent item.

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. When the control sorts a column the position for each item can be changed. Use the handle of the item to identify an item. The ItemPosition property is not available if the control is running in the virtual mode. Use the SortChildren method to sort the child items. Use the SortOrder property to sort a column. Use the NextVisibleItem property to enumerate items as they are displayed. The SortableItem property specifies whether the item keeps its position after sorting.

property Items.ItemStrikeOut(Item as HITEM) as Boolean

Retrieves or sets the StrikeOut property of the Font object used to paint the item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Boolean	A boolean expression that indicates whether the item uses strikeout font attribute to paint it.

If the ItemStrikeOut property is True, the cell's font is displayed with a horizontal line through it. Use ItemBold, ItemItalic, ItemUnderline or ItemStrikeOut property to apply different font attributes to the item. Use the CellItalic, CellUnderline, CellBold or CellStrikeOut property to apply different font attributes to the cell. Use the CellValueFormat 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 Grid1_SelectionChanged()
   If Not (h = 0) Then Grid1.Items.ItemStrikeOut(h) = False
   h = Grid1.Items.SelectedItem()
   Grid1.Items.ItemStrikeOut(h) = True
End Sub
```

The following VB sample draws a horizontal line through the focused item:

```
With Grid1.Items
   .ItemStrikeOut(.FocusItem) = True
End With
```

The following C++ sample draws a horizontal line through the focused item:

```
#include "Items.h"
CItems items = m_grid.GetItems();
items.SetItemStrikeOut( items.GetFocusItem() , TRUE );
```

The following C# sample draws a horizontal line through the focused item:

```
axGrid1.Items.set_ItemStrikeOut(axGrid1.Items.FocusItem, true);
```

The following VB.NET sample draws a horizontal line through the focused item:

```
With AxGrid1.Items
   .ItemStrikeOut(.FocusItem) = True
End With
```

The following VFP sample draws a horizontal line through the focused item:

```
with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   .ItemStrikeOut( 0 ) = .t.
endwith
```

property Items.ItemToIndex (Item as HITEM) as Long

Retrieves the index of an item in the Items collection, given its handle.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Long	A long expression that indicates the index of 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.ItemToVirtual (Item as HITEM) as Long

Gets the index of the virtual item giving the handle of the item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Long	A long expression that indicates the index of the virtual item.

The ItemToVirtual property converts the handle of the item to the index of the virtual item/record. The ItemToVirtual property has effect only if the control is running in the virtual mode. Use the VirtualToItem property to get the handle of the item giving the index of the virtual item/record.

The following VB sample notifies the n object that the user changes the date in the control:

```
Private Sub Grid1_Change(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As Long,
newValue As Variant)
  With Grid1.Items
    n.Change .ItemToVirtual(Item), ColIndex, newValue
  End With
End Sub
```

property Items.ItemUnderline(Item as HITEM) as Boolean

Retrieves or sets the Underline property of the Font object used to paint the item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
Boolean	A boolean expression that indicates if the item is underlined or not. True if the item is underlined, False, if the item is not 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 CellValueFormat 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 Grid1_SelectionChanged()
   If Not (h = 0) Then Grid1.Items.ItemUnderline(h) = False
   h = Grid1.Items.SelectedItem()
   Grid1.Items.ItemUnderline(h) = True
End Sub
```

The following VB sample underlines the focused item:

```
With Grid1.Items
   .ItemUnderline(.FocusItem) = True
End With
```

The following C++ sample underlines the focused item:

```
#include "Items.h"
CItems items = m_grid.GetItems();
items.SetItemUnderline( items.GetFocusItem() , TRUE );
```

The following C# sample underlines the focused item:

```
axGrid1.Items.set_ItemUnderline(axGrid1.Items.FocusItem, true);
```

The following VB.NET sample underlines the focused item:

```
With AxGrid1.Items
   .ItemUnderline(.FocusItem) = True
End With
```

The following VFP sample underlines the focused item:

```
with thisform.Grid1.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 that was previously created using InsertControlItem property.
Long	A long expression that indicates the item's width.

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. Use the CellWidth property to specify the width of the cell, when it contains inner cells. Use the SplitCell property to split a cell.

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 property.

Type	Description
Item as HITEM	A long expression that indicates the item's handle that was previously created by InsertControlItem property.
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.

Type	Description
Item as HITEM	A long expression that indicates the item's handle that was previously created by InsertControlItem property.
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 ActiveX controls require additional window styles to be added to the conatiner window. For instance, the Web Brower added by the Grid1.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 to insert a Web control use the following lines:

```
Dim hWeb As HITEM
hWeb = Grid1.Items.InsertControlItem(, "https://www.exontrol.com")
Grid1.Items.ItemHeight(hWeb) = 196
```

Next step is adding the AddItem event handler:

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
   If (Grid1.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
```

```
    Grid1.Items.ItemWindowHostCreateStyle(Item) =
Grid1.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 the item's handle that indicates the last visible item.

The LastVisibleItem property retrieves the handle for the last visible item. To get the first visible item use FirstVisibleItem property. 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 LastVisibleItem(False) property gets the handle of the last visible item that's not a partial item. The LastVisibleItem(True) property gets the handle of the last visible item no matter if it is partially visible or not.

The following VB sample enumerates all visible items:

```
Private Sub VisItems(ByVal c As EXGRIDLibCtl.Grid)
   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_grid.GetItems();
long hItem = items.GetFirstVisibleItem();
while ( hItem )
{
   OutputDebugString( V2S( &items.GetCellValue( COleVariant( hItem ), COleVariant(
```

```
long(0) ) ) ) );
    hItem = items.GetNextVisibleItem( hItem );
}
```

The following C# sample enumerates all visible items:

```
EXGRIDLib.Items items = axGrid1.Items;
int hItem = items.FirstVisibleItem;
while (hItem != 0)
{
    object strCaption = items.get_CellValue(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 AxGrid1.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.Grid1.Items
    .DefaultItem = .FirstVisibleItem
    do while ( .DefaultItem <> 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.

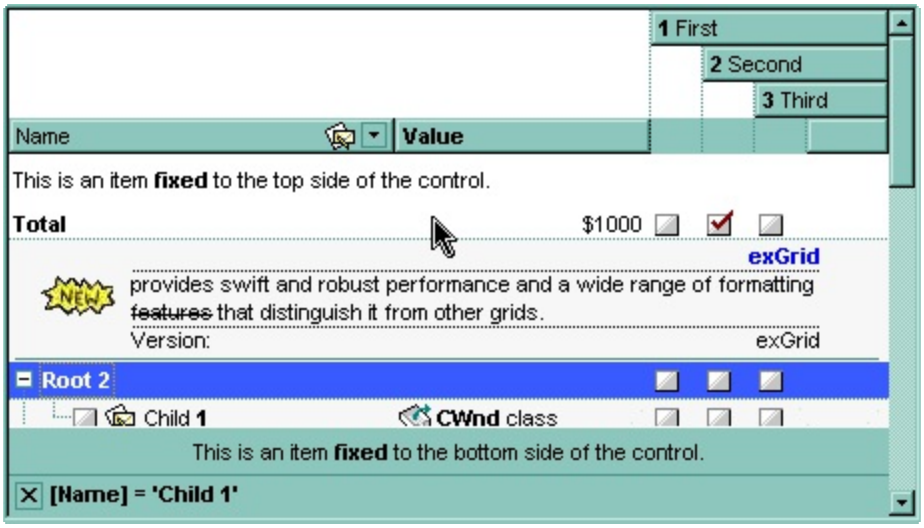
Type	Description
Alignment as VAlignmentEnum	A VAlignmentEnum expression that indicates whether the locked item requested is on the top or bottom side of the control.
Index as Long	A long expression that indicates the position of item being requested.
HITEM	A long expression that indicates the handle of the locked item

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 CellValue 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 in the same item. Use the MergeCells method to combine one or more cells in a single cell.



The following VB sample displays some locked items to the top and bottom side of the control (before running the sample, please make sure that the control contains some columns:

```
With Grid1
   .BeginUpdate
   .DrawGridLines = exHLines
   .ShowLockedItems = True
   If .Columns.Count > 0 Then
```

```
    With .Items
      Dim h As EXGRIDLibCtl.HITEM, a As EXGRIDLibCtl.VAlignmentEnum
      a = exTop
      .LockedItemCount(a) = 2
      h = .LockedItem(a, 0)
      .CellValue(h, 0) = "This is an item <b>fixed</b> to the top side of the control."
      .CellValueFormat(h, 0) = exHTML
      .ItemHeight(h) = 24
      .ItemDivider(h) = 0
      .ItemDividerLine(h) = EmptyLine
      h = .LockedItem(a, 1)
      .CellValue(h, 0) = "Total"
      .CellBold(h, 0) = True
      .CellValue(h, 1) = "$1000"
      .CellEditorVisible(h, 1) = False
      .CellHAlignment(h, 1) = RightAlignment

      a = exBottom
      .LockedItemCount(a) = 1
      h = .LockedItem(a, 0)
      .CellValue(h, 0) = "This is an item <b>fixed</b> to the bottom side of the control."
      .CellValueFormat(h, 0) = exHTML
      .CellHAlignment(h, 0) = CenterAlignment
      .ItemHeight(h) = 24
      .ItemDivider(h) = 0
      .ItemDividerLine(h) = EmptyLine
      .ItemBackColor(h) = Grid1.BackColorHeader

    End With
  End If
  .EndUpdate
End With
```

The following C++ sample adds an item that's locked to the top side of the control:

```
#include "Items.h"
m_grid.BeginUpdate();
```

```
CItems items = m_grid.GetItems();
items.SetLockedItemCount( 0 /*exTop*/, 1);
long i = items.GetLockedItem( 0 /*exTop*/, 0 );
COleVariant vtItem(i), vtColumn( long(0) );
items.SetCellValue( vtItem, vtColumn, COleVariant( "<b>locked</b> item" ) );
items.SetCellValueFormat( vtItem, vtColumn, 1/*exHTML*/ );
m_grid.EndUpdate();
```

The following VB.NET sample adds an item that's locked to the top side of the control:

```
With AxGrid1
   .BeginUpdate()
   With .Items
      .LockedItemCount(EXGRIDLib.VAlignmentEnum.exTop) = 1
      Dim i As Integer
      i = .LockedItem(EXGRIDLib.VAlignmentEnum.exTop, 0)
      .CellValue(i, 0) = "<b>locked</b> item"
      .CellValueFormat(i, 0) = EXGRIDLib.CaptionFormatEnum.exHTML
   End With
   .EndUpdate()
End With
```

The following C# sample adds an item that's locked to the top side of the control:

```
axGrid1.BeginUpdate();
EXGRIDLib.Items items = axGrid1.Items;
items.set_LockedItemCount(EXGRIDLib.VAlignmentEnum.exTop, 1);
int i = items.get_LockedItem(EXGRIDLib.VAlignmentEnum.exTop, 0);
items.set_CellValue(i, 0, "<b>locked</b> item");
items.set_CellValueFormat(i, 0, EXGRIDLib.CaptionFormatEnum.exHTML);
axGrid1.EndUpdate();
```

The following VFP sample adds an item that's locked to the top side of the control:

```
with thisform.Grid1
   .BeginUpdate()
      With .Items
         .LockedItemCount(0) = 1
         .DefaultItem = .LockedItem(0, 0)
```

```
        .CellValue(0, 0) = "<b>locked</b> item"
        .CellValueFormat(0, 0) = 1 && EXGRIDLib.ValueFormatEnum.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:

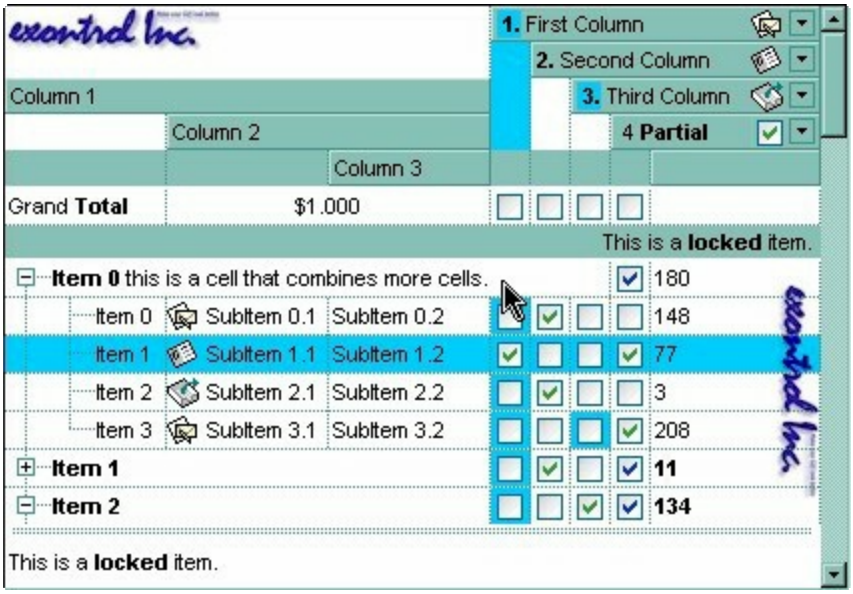
- 0, the control displays/contains no items, and no filter is applied to any column
- -1, the control displays no items, and there is a filter applied (no match found)
- positive number, indicates the number of items within the control (ItemCount property)
- negative number, the absolute value minus 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.

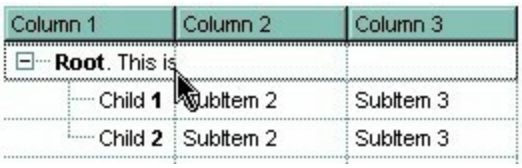
The MergeCells method combines two or more cells into one cell. The data in the **first specified cell** is displayed in the new larger cell. All the other cells' data is not lost. Use the CellMerge property to merge or unmerge a cell with another cell in the same item. Use the ItemDivider property to display a single cell in the entire item. Use the UnmergeCells method to unmerge the merged cells. Use the CellValue property to specify the cell's value. 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.



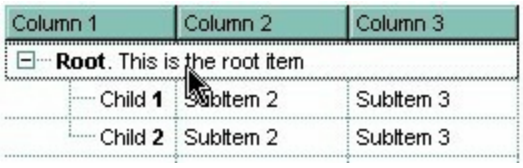
The following samples adds three columns, a root item and two child items:

```
With Grid1
   .BeginUpdate
      .MarkSearchColumn = False
      .DrawGridLines = exAllLines
      .LinesAtRoot = exLinesAtRoot
      With .Columns.Add("Column 1")
         .Def(exCellValueFormat) = 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:



The following VB sample merges the first three cells:

```
With Grid1.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_grid.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 AxGrid1.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:

```
EXGRIDLib.Items items = axGrid1.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.Grid1.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

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

```
 With Grid1
   With .Items
      .CellMerge(.RootItem(0), 0) = Array(1, 2)
   End With
End With
```

```
With Grid1
   .BeginUpdate
   With .Items
      Dim r As Long
      r = .RootItem(0)
      .CellMerge(r, 0) = 1
      .CellMerge(r, 0) = 2
   End With
   .EndUpdate
End With
```

```
With Grid1
   .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 Grid1
    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 Grid1
    With .Items
        Dim r As Long
        r = .RootItem(0)
        .MergeCells Array(.ItemCell(r, 0), .ItemCell(r, 1), .ItemCell(r, 2))
    End With
End With
```

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 next sibling item's handle.

The NextSiblingItem property retrieves the next sibling of the item in the parent's child list. Use ItemChild and NextSiblingItem properties to enumerate the collection of child items. Use the FirstVisibleItem property to get the first visible item in the control's client area. The NextVisibleItem property retrieves the handle of next visible item.

The following VB function recursively enumerates the item and all its child items:

```
Sub RecItem(ByVal c As EXGRIDLibCtl.Grid, 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( CGrid* pGrid, long hItem )
{
   COleVariant vtColumn( (long)0 );
   if ( hItem )
   {
      CItems items = pGrid->GetItems();
```

```
        CString strCaption = V2S( &items.GetCellValue( COleVariant( hItem ), vtColumn ) ),
strOutput;
        strOutput.Format( "Cell: '%s'\n", strCaption );
        OutputDebugString( strOutput );

        long hChild = items.GetItemChild( hItem );
        while ( hChild )
        {
            RecItem( pGrid, 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 AxEXGRIDLib.AxGrid, 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(AxEXGRIDLib.AxGrid  grid, int hItem)
{
    if (hItem != 0)
    {
        EXGRIDLib.Items items = grid.Items;
        object caption = items.get_CellValue( hItem, 0 );
```

```
        System.Diagnostics.Debug.WriteLine(caption != null ? caption.ToString() : "");

        int hChild = items.get_ItemChild(hItem);
        while (hChild != 0)
        {
            RecItem(grid, 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.Grid1
    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 FirstVisibleItem property to get the first visible item in the control's client area. Use the RootItem property to get the first visible item in the list. The NextSiblingItem property retrieves the next sibling of the item in the parent's child list. Use the IsItemVisible property to check whether an item fits the control's client area. Use the ItemPosition property to change the position of the item. Use the SortOrder property to sort a column.

The following VB sample enumerates all visible items:

```
Private Sub VisItems(ByVal c As EXGRIDLibCtl.Grid)
  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 VB sample enumerates all cells in the control, as they are listed:

```
With Grid1
  Dim nCols As Long
  nCols = .Columns.Count
  With .Items
    Dim h As HITEM
    h = .RootItem(0)
    While Not h = 0
      Dim i As Long
```

```
        For i = 0 To nCols - 1
            Debug.Print .CellValue(h, i)
        Next
        h = .NextVisibleItem(h)
    Wend
  End With
End With
```

The following C++ sample enumerates all visible items:

```
#include "Items.h"
CItems items = m_grid.GetItems();
long hItem = items.GetFirstVisibleItem();
while ( hItem )
{
   OutputDebugString( V2S( &items.GetCellValue( COleVariant( hItem ), COleVariant(
long(0) ) ) ) );
   hItem = items.GetNextVisibleItem( hItem );
}
```

The following C# sample enumerates all visible items:

```
EXGRIDLib.Items items = axGrid1.Items;
int hItem = items.FirstVisibleItem;
while (hItem != 0)
{
   object strCaption = items.get_CellValue(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 AxGrid1.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.Grid1.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. Use the RootItem property to get the first visible item in the list.

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. Use the RootItem property to get the first visible item in the list.

method Items.RemoveAllItems ()

Removes all items from the control.

Type	Description

The RemoveAllItems method remove all items in the control. The Clear method of Columns object clears the Items collection too. Use the RemoveItem method to remove an item from the control. The RemoveAllItems method is not available if the control is running in the virtual mode.

method Items.RemoveItem (Item as HITEM)

Removes the given item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle being removed.

The RemoveItem method removes an item. The RemoveItem method does not remove the item, if it contains child items. 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 RemoveItem method is not available if the control is running in the virtual mode. The RemoveSelection method removes the selected items (including the descendents).

The following VB sample removes recursively an item:

```
Private Sub RemoveItemRec(ByVal t As EXGRIDLibCtl.Grid, ByVal h As HITEM)
  t.BeginUpdate
    With t.Items
      If (.ChildCount(h) > 0) Then
        Dim hChild As HITEM
        hChild = .ItemChild(h)
        While (hChild <> 0)
          Dim hNext As HITEM
          hNext = .NextSiblingItem(hChild)
          RemoveItemRec t, hChild
          hChild = hNext
        Wend
      End If
    .RemoveItem h
  t.EndUpdate
  End With
End Sub
```

The following C++ sample removes recursively an item:

```
void RemoveItemRec( CGrid* pGrid, long hItem )
```

```
{
   if ( hItem )
   {
      pGrid->BeginUpdate();
      CItems items = pGrid->GetItems();
      long hChild = items.GetItemChild( hItem );
      while ( hChild )
      {
         long nNext = items.GetNextSiblingItem( hChild );
         RemoveItemRec( pGrid, hChild );
         hChild = nNext;
      }
      items.RemoveItem( hItem );
      pGrid->EndUpdate();
   }
}
```

The following VB.NET sample removes recursively an item:

```
Shared Sub RemoveItemRec(ByVal t As AxEXGRIDLib.AxGrid, 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(AxEXGRIDLib.AxGrid grid, int hItem)
```

```
{
   if (hItem != 0)
   {
      EXGRIDLib.Items items = grid.Items;
      grid.BeginUpdate();
      int hChild = items.get_ItemChild(hItem);
      while (hChild != 0)
      {
         int hNext = items.get_NextSiblingItem(hChild);
         RemoveItemRec(grid, hChild);
         hChild = hNext;
      }
      items.RemoveItem(hItem);
      grid.EndUpdate();
   }
}
```

The following VFP sample removes recursively an item (removeitemrec method):

```
LPARAMETERS h

with thisform.Grid1
   If ( h != 0 ) Then
      .BeginUpdate()
      local hChild
      With .Items
         hChild = .ItemChild(h)
         do While (hChild != 0)
            local hNext
            hNext = .NextSiblingItem(hChild)
            thisform.removeitemrec(hChild)
            hChild = hNext
         enddo
         .RemoveItem( h )
      EndWith
      .EndUpdate()
   EndIf
```

endwith

method Items.RemoveSelection ()

Removes the selected items (including the descendents).

Type	Description

The RemoveSelection method removes the selected items (including the descendents). The RemoveItem method removes a specific item (if it has not child items). The UnselectAll method unselects all items in the list.

property Items.RootCount as Long

Retrieves the number of root objects in the Items collection.

Type	Description
Long	A long value that indicates the count of root items into Items collection.

A root item is an item that has no parent (ItemParent() = 0). Use the RootItem property of the Items object to enumerates the root items. Use the AddItem to add root items to the control. Use the InsertItem method to insert child items.

The following VB sample enumerates all root items:

```
Dim i As Long, n As Long
With Grid1.Items
   n = .RootCount
   For i = 0 To n - 1
      Debug.Print .CellValue(.RootItem(i), 0)
   Next
End With
```

The following C++ sample enumerates all root items:

```
#include "Items.h"
CItems items = m_grid.GetItems();
for ( long i = 0 ; i < items.GetRootCount(); i++ )
{
   COleVariant vtItem( items.GetRootItem(i) ), vtColumn( long(0) );
   OutputDebugString( V2S( &items.GetCellValue( vtItem, vtColumn ) ) );
}
```

where the V2S function converts a VARIANT expression to a string expression and looks like:

```
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;
}
```

if you are using MFC, or

```
static string V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        CComVariant vt;
        if ( SUCCEEDED( vt.ChangeType( VT_BSTR, pv ) ) )
        {
            USES_CONVERSION;
            return OLE2T(V_BSTR( &vt ));
        }
    }
    return szDefault;
}
```

if you are using STL.

The following VB.NET sample enumerates all root items:

```
With AxGrid1.Items
    Dim i As Integer
    For i = 0 To .RootCount - 1
        Debug.Print(.CellValue(.RootItem(i), 0))
    Next
End With
```

The following C# sample enumerates all root items:

```
for (int i = 0; i < axGrid1.Items.RootCount; i++)
{
    object strValue = axGrid1.Items.get_CellValue(axGrid1.Items.get_RootItem(i), 0);
    System.Diagnostics.Debug.WriteLine(strValue != null ? strValue.ToString() : "");
}
```

The following VFP sample enumerates all root items:

```
with thisform.Grid1.Items
    local i
    for i = 0 to .RootCount - 1
        .DefaultItem = .RootItem(i)
        wait window nowait .CellValue(0,0)
    next
endwith
```

property Items.RootItem ([Position as Long]) as HITEM

Retrieves the handle of the root item given its index in the root items collection.

Type	Description
Position as Long	A long value that indicates the index of the root item.
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. Use the FirstVisibleItem property to get the first visible item in the control's client area. The NextVisibleItem property retrieves the handle of next visible item. The NextSiblingItem property retrieves the next sibling of the item in the parent's child list. Use the RootItem property to get the first visible item in the list. If you need to enumerate the items as they are added, you may use the ItemByIndex property.

The following VB sample enumerates all root items. This sample can be used to list all your items as they are displayed (sorted), when the control displays a plain list of items. A plain list of items is composed by items that do not have child items.

```
Dim i As Long, n As Long
With Grid1.Items
   n = .RootCount
   For i = 0 To n - 1
      Debug.Print .CellValue(.RootItem(i), 0)
   Next
End With
```

The following VB sample enumerates all cells in the control, as they are listed:

```
With Grid1
   Dim nCols As Long
   nCols = .Columns.Count
   With .Items
      Dim h As HITEM
      h = .RootItem(0)
      While Not h = 0
         Dim i As Long
         For i = 0 To nCols - 1
```

```
            Debug.Print .CellValue(h, i)
        Next
        h = .NextVisibleItem(h)
    Wend
  End With
End With
```

The following C++ sample enumerates all root items:

```
#include "Items.h"
CItems items = m_grid.GetItems();
for ( long i = 0 ; i < items.GetRootCount(); i++ )
{
   COleVariant vtItem( items.GetRootItem(i) ), vtColumn( long(0) );
   OutputDebugString( V2S( &items.GetCellValue( vtItem, vtColumn ) ) );
}
```

where the V2S function converts a VARIANT expression to a string expression and looks like:

```
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;
}
```

if you are using MFC, or

```
static string V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
   if ( pv )
```

```
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        CComVariant vt;
        if ( SUCCEEDED( vt.ChangeType( VT_BSTR, pv ) ) )
        {
            USES_CONVERSION;
            return OLE2T(V_BSTR( &vt ));
        }
    }
    return szDefault;
}
```

if you are using STL.

The following VB.NET sample enumerates all root items:

```
With AxGrid1.Items
    Dim i As Integer
    For i = 0 To .RootCount - 1
        Debug.Print(.CellValue(.RootItem(i), 0))
    Next
End With
```

The following C# sample enumerates all root items:

```
for (int i = 0; i < axGrid1.Items.RootCount; i++)
{
    object strValue = axGrid1.Items.get_CellValue(axGrid1.Items.get_RootItem(i), 0);
    System.Diagnostics.Debug.WriteLine(strValue != null ? strValue.ToString() : "");
}
```

The following VFP sample enumerates all root items:

```
with thisform.Grid1.Items
    local i
    for i = 0 to .RootCount - 1
        .DefaultItem = .RootItem(i)
```

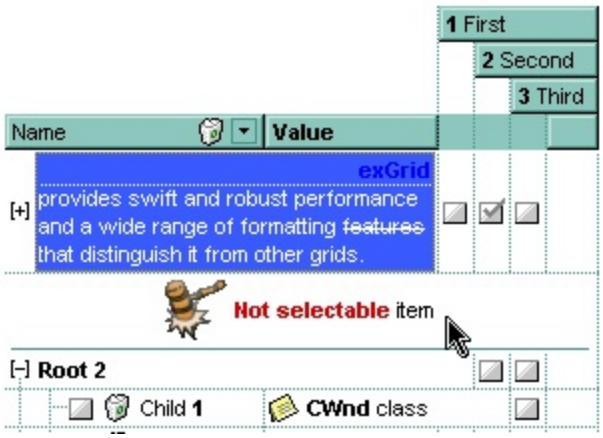
```
    wait window nowait .CellValue(0,0)
  next
endwith
```

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. Use the ItemHeight property and SelectableItem property to hide an item. The SortableItem property specifies whether the item keeps its position after sorting.



The following VB sample makes not selectable the first visible item:

```
With Grid1.Items
  .SelectableItem(.FirstVisibleItem) = False
```

```
End With
```

The following C++ sample makes not selectable the first visible item:

```
CItems items = m_grid.GetItems();
items.SetSelectableItem( items.GetFirstVisibleItem(), FALSE );
```

The following VB.NET sample makes not selectable the first visible item:

```
With AxGrid1.Items
   .SelectableItem(.FirstVisibleItem) = False
End With
```

The following C# sample makes not selectable the first visible item:

```
axGrid1.Items.set_SelectableItem(axGrid1.Items.FirstVisibleItem, false);
```

The following VFP sample makes not selectable the first visible item:

```
with thisform.Grid1.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. The SelectPos property selects/unselects items by position. The Selection property selects/unselects items by index.

property Items.SelectCount as Long

Counts the number of items that are selected in the control.

Type	Description
Long	A long expression that identifies the number of selected items.

The SelectCount property counts the selected items in the control. The control supports single or multiple selection. Use SingleSel property of the control to enable multiple selection. Use the SelectedItem property to retrieve the handle of the selected item(s). Use the SelBackColor property to indicate the background color for selected items. Use the SelForeColor property to specify the foreground color for selected items. The FocusItem property specifies the handle of the focused item. For instance, if the control supports single selection the FocusItem property retrieves the handle of the selected item too. Use the FullRowSelect property to specify how the user can select the cells or items using the mouse. Use the SelectItem property to programmatically select an item giving its handle. The control fires the SelectionChanged event when user changes the selection in the control. Use the SelectableItem property to specify whether the user can select an item.

The following VB sample enumerates all selected items in the control:

```
Dim i As Long, j As Long, nCols As Long, nSels As Long
nCols = Grid1.Columns.Count
With Grid1.Items
   nSels = .SelectCount
   For i = 0 To nSels - 1
      Dim s As String
      For j = 0 To nCols - 1
         s = s + .CellValue(.SelectedItem(i), j) + Chr(9)
      Next
      Debug.Print s
   Next
End With
```

The following VB sample unselects all items in the control:

```
With Grid1
   .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_grid.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.GetCellValue( COleVariant( h ), COleVariant(
(long)0 ) ) );
        CString str = V_BSTR( &vtString );
        MessageBox( str );
    }
}
```

The following C++ sample unselects all items in the control:

```
m_grid.BeginUpdate();
CItems items = m_grid.GetItems();
while ( items.GetSelectCount() )
    items.SetSelectItem( items.GetSelectedItem( 0 ), FALSE );
m_grid.EndUpdate();
```

The following VB.NET sample enumerates the selected items:

```
With AxGrid1.Items
    Dim nCols As Integer = AxGrid1.Columns.Count, i As Integer
    For i = 0 To .SelectCount - 1
        Debug.Print(.CellValue(.SelectedItem(i), 0))
    Next
```

```
End With
```

The following VB.NET sample unselects all items in the control:

```
With AxGrid1
   .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 < axGrid1.Items.SelectCount; i++)
{
   object strCaption = axGrid1.Items.get_CellValue(axGrid1.Items.get_SelectedItem(i), 0);
   System.Diagnostics.Debug.WriteLine(strCaption != null ? strCaption.ToString() : "");
}
```

The following C# sample unselects all items in the control:

```
axGrid1.BeginUpdate();
EXGRIDLib.Items items = axGrid1.Items;
while (items.SelectCount != 0)
   items.set_SelectItem(items.get_SelectedItem(0), false);
axGrid1.EndUpdate();
```

The following VFP sample enumerates the selected items:

```
with thisform.Grid1.Items
   local i
   for i = 0 to .SelectCount - 1
      .DefaultItem = .SelectedItem(i)
      wait window nowait .CellValue(0,0)
   next
endwith
```

The following VFP sample unselects all items in the control:

```
With thisform.Grid1
   .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 the selected items collection.

Type	Description
Index as Long	Identifies the index of the selected item into selected items collection. if it is missing, 0 is used.
HITEM	A long expression that indicates the handle of the selected item.

The SelectedItem property gets the handle of the items being selected. If the control supports multiple selection, you can use the SelectCount property to find out how many items are selected in the control. Use the SingleSel property to enable single or multiple selection. If the control supports single selection only a single item can be selected at runtime. Use the SingleSel property to specify whether the control supports single or multiple selection. If the control supports single selection, the FocusItem and SelectedItem property gets the handle of the selected/focused item, that's the same. Use the SelectItem property to specify whether an item is selected or not. The control fires the SelectionChanged event when user changes the selection in the control. Use the SelForeColor and SelBackColor properties to specify colors for selected items. Use the SelectableItem property to specify whether the user can select an item.

The following VB sample prints the value of the selected/focused cell: Debug.Print Grid1.Items.CellValue(Grid1.Items.FocusItem(), 0).

The following VB sample draws italic the selected item, when selection is changed:

```
Private Sub Grid1_SelectionChanged()
  If Not (h = 0) Then Grid1.Items.ItemItalic(h) = False
  h = Grid1.Items.SelectedItem()
  Grid1.Items.ItemItalic(h) = True
End Sub
```

The following VB sample enumerates all selected items in the control:

```
Dim i As Long, j As Long, nCols As Long, nSels As Long
nCols = Grid1.Columns.Count
With Grid1.Items
  nSels = .SelectCount
  For i = 0 To nSels - 1
    Dim s As String
```

```
        For j = 0 To nCols - 1
            s = s + .CellValue(.SelectedItem(i), j) + Chr(9)
        Next
        Debug.Print s
    Next
End With
```

The following VB sample unselects all items in the control:

```
With Grid1
    .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_grid.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.GetCellValue( COleVariant( h ), COleVariant(
(long)0 ) ) );
        CString str = V_BSTR( &vtString );
        MessageBox( str );
    }
}
```

The following C++ sample unselects all items in the control:

```
m_grid.BeginUpdate();
```

```
CItems items = m_grid.GetItems();
while ( items.GetSelectCount() )
   items.SetSelectItem( items.GetSelectedItem( 0 ), FALSE );
m_grid.EndUpdate();
```

The following VB.NET sample enumerates the selected items:

```
With AxGrid1.Items
   Dim nCols As Integer = AxGrid1.Columns.Count, i As Integer
   For i = 0 To .SelectCount - 1
      Debug.Print(.CellValue(.SelectedItem(i), 0))
   Next
End With
```

The following VB.NET sample unselects all items in the control:

```
With AxGrid1
   .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 < axGrid1.Items.SelectCount; i++)
{
   object strCaption = axGrid1.Items.get_CellValue(axGrid1.Items.get_SelectedItem(i), 0);
   System.Diagnostics.Debug.WriteLine(strCaption != null ? strCaption.ToString() : "");
}
```

The following C# sample unselects all items in the control:

```
axGrid1.BeginUpdate();
EXGRIDLib.Items items = axGrid1.Items;
while (items.SelectCount != 0)
```

```
    items.set_SelectItem(items.get_SelectedItem(0), false);
axGrid1.EndUpdate();
```

The following VFP sample enumerates the selected items:

```
with thisform.Grid1.Items
    local i
    for i = 0 to .SelectCount - 1
        .DefaultItem = .SelectedItem(i)
        wait window nowait .CellValue(0,0)
    next
endwith
```

The following VFP sample unselects all items in the control:

```
With thisform.Grid1
    .BeginUpdate()
    with .Items
        do while ( .SelectCount() # 0 )
            .DefaultItem = .SelectedItem(0)
            .SelectItem(0) = .f.
        enddo
    endwith
    .EndUpdate()
EndWith
```

property Items.Selection as Variant

Selects items by index.

Type	Description
Variant	A long expression that indicates the index of item being selected, if the SingleSel property is True, or a safe array that holds a collection of index of items being selected, if the SingleSel property is False.

The Selection property selects/unselects items by index. Use the SelectItem property to select an item giving its handle. The ItemPosition property gives the relative position, or the position of the item in the child items collection. Use the SelectPos property to select items by position. The SelectPos property selects an item giving its general position.

The SingleSel property specifies whether the control supports single or multiple-selection. Based on the SingleSel property the Selection value is:

- of long type, if the SingleSel property is True (by default). For instance Selection = 0, indicates that the control selects the item with the index 0.
- a safe array of VARIANT, if the SingleSel property is False. For instance Selection = Array(0,1), indicates that the control selects the item with the index 0 and 1.

The following VB sample selects the item with the index 0 in the control / SingleSel property is True (by default):

```
Grid1.Items.Selection = 0
```

The following VB sample selects the item with the index 0 and 1 in the control / SingleSel property is False:

```
Grid1.Items.Selection = Array(0, 1)
```

The following C++ sample selects the item with the index 0 in the control / SingleSel property is True (by default):

```
m_grid.GetItems().SetSelection( COleVariant( long(0) ) );
```

The following C++ sample selects the item with the index 0 in the control / SingleSel property is False:

```
CArray<long> a;
a.Add( 0 );
```

```
a.Add( 1 );
m_spExGrid->Items->Selection = CreateSafeArray( a );
```

where the CreateSafeArray looks as:

```
CComVariant CreateSafeArray(CArray<long>& a)
{
   CComVariant vtResult;

   long nCount = a.GetCount();
   if ( SAFEARRAY* pArray = SafeArrayCreateVector( VT_VARIANT, 0, nCount ) )
   {
      LPVOID pData = NULL;
      SafeArrayAccessData( pArray, &pData );
      VARIANT* p = (VARIANT*)pData;
      for ( long i = 0; i < nCount; p++, i++ )
      {
         ZeroMemory( p, sizeof( VARIANT ) );
         V_VT( p ) = VT_I4;
         V_I4( p ) = a.GetAt( i );
      }
      SafeArrayUnaccessData( pArray );

      V_VT( &vtResult ) = VT_ARRAY | VT_VARIANT;
      V_ARRAY( &vtResult ) = pArray;
   }
   return vtResult;
}
```

The following VB.NET sample selects the item with the index 0 in the control / SingleSel property is True (by default):

```
With AxGrid1.Items
   .Selection = 0
End With
```

The following C# sample selects the item with the index 0 in the control / SingleSel property is True (by default):

```
axGrid1.Items.Selection = 0;
```

The following VFP sample selects the item with the index 0 in the control / SingleSel property is True (by default):

```
with thisform.Grid1.Items
   .Selection = 0
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 being selected.
Boolean	A boolean expression that indicates the item's state. True if the item is selected, and False if the item is not selected.

Use the SelectItem property to programmatically select an item. The SelectItem property indicates whether an item is selected or not selected, giving its handle. Use the SelectedItem property to get the selected items, giving their indexes. The control fires SelectionChanged event when the user changes the selection. The SelectCount property counts the selected items in the control, when the control supports multiple selection. Use the SingleSel property to specify whether the control supports single or multiple selection. If the SingleSel property is True, the user can select a single item only. Use the FullRowSelect property to specify how the user can select the cells or items using the mouse. Use the SelectColumnIndex or SelectColumInner property to retrieve the index of selected column, and the index of the inner cell being selected. The FocusItem property specifies the handle of the focused item. The control can have only a single focused item. If the control supports single selection, the FocusItem property gets the selected item too. Use the EnsureVisibleItem property to ensure that an item is visible. Use the SelBackColor property to indicate the background color for selected items. Use the SelForeColor property to specify the foreground color for selected items. The SelectPos property selects/unselects items by position. The Selection property selects/unselects items by index.

The following VB sample selects the first created item:
Grid1.Items.SelectItem(Grid1.Items(0)) = True

The following VB sample displays the selected item from the cursor (SingleSel property is True, and the control contains **NO** inner cells, SplitCell method):

```
Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
  Dim h As HITEM, c As Long, hit As HitTestInfoEnum
  With Grid1
    h = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
    If Not h = 0 Then
      If (.Items.SelectItem(h)) Then
```

```
            Debug.Print "The '" & .Items.CellCaption(h, c) & "' item is selected"
        End If
    End If
  End With
End Sub
```

The following VB sample displays the selected item from the cursor, (SingleSel property is True, and the control contains inner cells, SplitCell method)

```
Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  Dim h As HITEM, c As Long, hit As HitTestInfoEnum
  With Grid1
    h = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
    If Not h = 0 Or Not c = 0 Then
        If h = 0 Then
            h = .Items.CellItem(.Items.CellParent(h, c))
        End If
      If (.Items.SelectItem(h)) Then
            Debug.Print "The '" & .Items.CellCaption(h, c) & "' item is selected"
      End If
    End If
  End With
End Sub
```

The following VB sample selects the item as user moves the cursor (SingleSel property is True, and the control contains NO inner cells, SplitCell method):

```
Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
  Dim h As HITEM, c As Long, hit As HitTestInfoEnum
  With Grid1
    h = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
    If Not h = 0 Then
      .Items.SelectItem(h) = True
    End If
  End With
End Sub
```

The following VB sample selects the item as user moves the cursor (SingleSel property is True, and the control contains inner cells, SplitCell method):

```
Private Sub Grid1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
   Dim h As HITEM, c As Long, hit As HitTestInfoEnum
   With Grid1
      h = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit)
      If Not h = 0 Or Not c = 0 Then
         If h = 0 Then
            h = .Items.CellItem(.Items.CellParent(h, c))
         End If
        .Items.SelectItem(h) = True
      End If
   End With
End Sub
```

The following VB sample selects the first visible item:

```
With Grid1.Items
   .SelectItem(.FirstVisibleItem) = True
End With
```

The following VB sample unselects all items in the control:

```
With Grid1
   .BeginUpdate
   With .Items
      While Not .SelectCount = 0
         .SelectItem(.SelectedItem(0)) = False
      Wend
   End With
   .EndUpdate
End With
```

The following C++ sample selects the first visible item:

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

```
items.SetSelectItem( items.GetFirstVisibleItem(), TRUE );
```

The following C++ sample unselects all items in the control:

```
m_grid.BeginUpdate();
CItems items = m_grid.GetItems();
while ( items.GetSelectCount() )
   items.SetSelectItem( items.GetSelectedItem( 0 ), FALSE );
m_grid.EndUpdate();
```

The following VB.NET sample selects the first visible item:

```
With AxGrid1.Items
   .SelectItem(.FirstVisibleItem) = True
End With
```

The following VB.NET sample unselects all items in the control:

```
With AxGrid1
   .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:

```
axGrid1.Items.set_SelectItem(axGrid1.Items.FirstVisibleItem, true);
```

The following C# sample unselects all items in the control:

```
axGrid1.BeginUpdate();
EXGRIDLib.Items items = axGrid1.Items;
while (items.SelectCount != 0)
   items.set_SelectItem(items.get_SelectedItem(0), false);
axGrid1.EndUpdate();
```

The following VFP sample selects the first visible item:

```
with thisform.Grid1.Items
   .DefaultItem = .FirstVisibleItem
   .SelectItem(0) = .t.
endwith
```

The following VFP sample unselects all items in the control:

```
With thisform.Grid1
   .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, if the SingleSel property is True, or a safe array that holds a collection of position of items being selected, if the SingleSel property is False.

Use the SelectPos property to select items by position. The SelectPos property selects an item giving its general position. Use the SelectItem property to select an item giving its handle. The ItemPosition property gives the relative position, or the position of the item in the child items collection. The Selection property selects/unselects items by index.

The SingleSel property specifies whether the control supports single or multiple-selection. Based on the SingleSel property the SelectPos value is:

- of long type, if the SingleSel property is True (by default). For instance SelectPos = 0, indicates that the control selects the item with the position 0 (first item).
- a safe array of VARIANT, if the SingleSel property is False. For instance SelectPos = Array(0,1), indicates that the control selects the item with the position 0 and 1.

The following VB sample selects the first item in the control:

```
Grid1.Items.SelectPos = 0
```

The following VB sample selects first two items:

```
Grid1.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 AxGrid1.Items
   .SelectPos = 0
End With
```

The following C# sample selects the first item in the control:

```
axGrid1.Items.SelectPos = 0;
```

The following VFP sample selects the first item in the control:

```
with thisform.Grid1.Items
    .SelectPos = 0
endwith
```

method Items.SetParent (Item as HITEM, NewParent as HITEM)

Changes the parent of the given item.

Type	Description
Item as HITEM	A long expression that indicates the item's handle.
NewParent as HITEM	A long expression that indicates the handle of the newly parent item.

Use the SetParent property to change the parent item at runtime. Use the InsertItem property to insert child items. Use 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: Grid1.Items.SetParent Grid1.Items(0), Grid1.Items(1). Use the ItemParent property to retrieve the parent of the item.

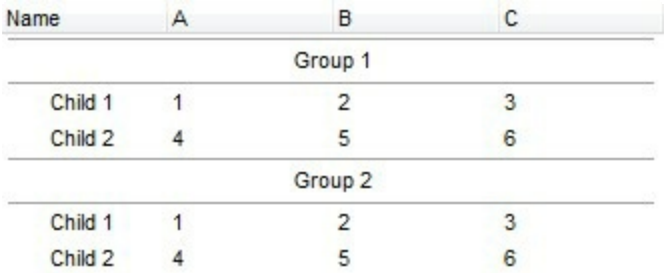
property Items.SortableItem(Item as HITEM) as Boolean

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)



The following screen shots shows the control when the column A is being sorted: (Group 1 and Group 2 keeps their original position after sorting)

method Items.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.
ColIndex as Variant	A long expression that indicates the column's index, a string expression that indicates the column's caption.
Ascending as Boolean	A boolean expression that defines the sort order. True means ascending, False means descending.

The SortChildren method will not recurse through the grid, only the immediate children of item will be sorted. After sort, the control ensures that the focused item fits the control's client area. Use the FocusItem property to retrieve the focused item. If your control acts like a simple list you can use the following line of code to sort ascending the list by first column: Grid1.Items.SortChildren 0, 0, True. To change the way how a column is sorted use SortType property of Column object. The SortChildren property doesn't display the sort icon on column's header. The control automatically sorts the children items when user clicks on column's header. Use the SortOnClick property to disable sorting columns by clicking in the columns header. Use the SortOrder property to get the column sorted, and to display the sorting icon in the column's header. The EnsureOnSort property prevents scrolling the control's content when the user sorts items. The SortableItem property specifies whether the item keeps its position after sorting. Use the AllowSort property to avoid sorting a column when the user clicks the column. The SortChildren method is not available if the control is running in the virtual mode.

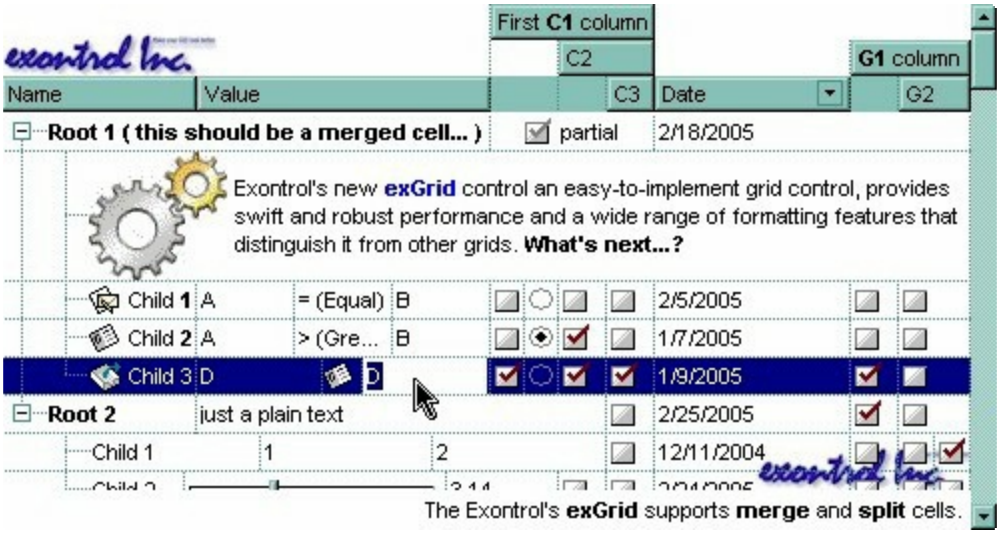
property Items.SplitCell ([Item as Variant], [ColIndex as Variant]) as Variant

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 CellValue property to assign a value 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. The SelectColumInner property indicates the index of the inner cell that has the focus (or it is selected). Use the SelectableItem property to specify the user can select an item. Include the exIncludeInnerCells flag in the FilterList property and so the drop down filter window lists the inner cells too.

("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 Grid1.Items
   Dim h As HITEM, f As HCELL
   h = .FirstVisibleItem
   f = .SplitCell(h, 0)
   .CellValue(, f) = "inner cell"
End With
```

The following VB sample splits a cell in three cells (Before running the following sample, please make sure that your control contains columns, and at least an item):

```
With Grid1.Items
   Dim h As HITEM, f As HCELL
   h = .FirstVisibleItem
   f = .SplitCell(h, 0)
   .CellValue(, f) = "inner cell 1"
   f = .SplitCell(, f)
   .CellValue(, f) = "inner cell 2"
End With
```

The following C++ sample splits the first visible cell in two cells:

```
#include "Items.h"
CItems items = m_grid.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 AxGrid1.Items
   Dim i As Object
   i = .SplitCell(.FirstVisibleItem, 0)
   .CellValue(Nothing, i) = "inner cell"
End With
```

The following C# sample splits the first visible cell in two cells:

```
EXGRIDLib.Items items = axGrid1.Items;
object i = items.get_SplitCell(items.FirstVisibleItem, 0);
items.set_CellValue(null, i, "inner cell");
```

The following VFP sample splits the first visible cell in two cells:

```
with thisform.Grid1.Items
   local i
   i = .SplitCell(.FirstVisibleItem,0)
   local s, crlf
   crlf = chr(13) + chr(10)
   s = "Items" + crlf
   s = s + "{" + crlf
   s = s + "CellValue(," + str(i) + ") = " + chr(34) + "inner cell" + chr(34) + crlf
   s = s + "}"
   thisform.Grid1.Template = s
endwith
```

method Items.StartBlockUndoRedo ()

Starts recording the UI operations as a block of undo/redo operations.

Type	Description

The StartBlockUndoRedo method starts recording the UI operations as a block on undo/redo operations (equivalent of StartBlockUndoRedo method of the control). The method has effect only if the AllowUndoRedo property is True. The EndBlockUndoRedo method collects all undo/redo operations since StartBlockUndoRedo method was called and add them to the undo/redo queue as a block. This way the next call on a Undo operation, the entire block is restored, so all UI operations are restored. For instance, if you have a procedure that moves several bars, and want all of them being grouped, you can use StartBlockUndoRedo to start recording the operations as a block, and call the EndBlockUndoRedo when procedure ends, so next call of an undo operation the bars are restored to their original position. The EndBlockUndoRedo method must be called the same number of times as the StartBlockUndoRedo method was called. For instance, if you have called the StartBlockUndoRedo twice the EndBlockUndoRedo method must be called twice too, and the collected operations are added to the control's queue of undo/redo operations at the end.

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 sample shows few methods to unmerge cells:

```
With Grid1
   With .Items
      .UnmergeCells .ItemCell(.RootItem(0), 0)
   End With
End With
```

```
With Grid1
   With .Items
      Dim r As Long
      r = .RootItem(0)
      .UnmergeCells Array(.ItemCell(r, 0), .ItemCell(r, 1))
   End With
End With
```

```
With Grid1
   .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 SelectPos property selects/unselects items by position. The Selection property selects/unselects items by index.

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 cell).

property Items.VirtualToItem (Index as Long) as HITEM

Gets the handle of the item giving the index of the virtual item.

Type	Description
Index as Long	A long expression that indicates the index of the virtual item.
HITEM	A long expression that indicates the handle of the item.

The VirtualToItem property converts the the index of the virtual item/record to the handle of the item. The VirtualToItem property scrolls the control's content to make sure that the virtual item is in the control's client area. The VirtualToItem property has effect only if the control is running in the virtual mode. Use the ItemToVirtual property to get the index of the virtual item based on the handle of the item.

The following sample VB notifies the n object that the user changes the data in the control:

```
Private Sub Grid1_Change(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As Long,
newValue As Variant)
   With Grid1.Items
      n.Change .ItemToVirtual(Item), ColIndex, newValue
   End With
End Sub
```

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 property. Also the UserEditorOleEvent event uses the same type of the object to hold information about an OLE event.

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

property OleEvent.CountParam as Long

Retrieves the count of the OLE event's arguments.

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

Use the CountParam property to count the parameters of an OLE event. Use the Name property to get the parameter name. Use the Param property to get the event's parameter. Use the Value property to specify the value of the parameter. The following VB sample enumerates the arguments of an OLE event when ItemOLEEvent or UserEditorOleEvent event is fired.

```
Private Sub Grid1_ItemOleEvent(ByVal Item As EXGRIDLibCtl.HITEM, ByVal Ev As
EXGRIDLibCtl.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 VB sample displays information about the fired event when the UserEditorOleEvent event occurs:

```
Private Sub Grid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Item As EXGRIDLibCtl.HITEM, ByVal
ColIndex As Long)
   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 <exgrid.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 OnItemOleEventGrid1(long Item, LPDISPATCH Ev)
{
    EXGRIDLib::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++ )
        {
            EXGRIDLib::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 EXGRIDLib namespace that include all objects and types of the control's TypeLibrary. In case your exgrid.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 C++ sample displays the event and its parameters when an user editor object fires an event:

```
void OnUserEditorOleEventGrid1(LPDISPATCH Object, LPDISPATCH Ev, BOOL FAR*
CloseEditor, long Item, long ColIndex)
{
    EXGRIDLib::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++ )
      {
        EXGRIDLib::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 following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
Private Sub AxGrid1_ItemOleEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_ItemOleEventEvent) Handles AxGrid1.ItemOleEvent
   Debug.WriteLine("Event's name: " & e.ev.Name)
   Dim i As Long
   For i = 0 To e.ev.CountParam - 1
      Dim eP As EXGRIDLib.OleEventParam
      eP = e.ev(i)
      Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
   Next
End Sub
```

The following VB.NET sample displays the event and its parameters when an user editor object fires an event:

```
Private Sub AxGrid1_UserEditorOleEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent) Handles AxGrid1.UserEditorOleEvent
   Debug.WriteLine("Event's name: " & e.ev.Name)
   Dim i As Long
   For i = 0 To e.ev.CountParam - 1
      Dim eP As EXGRIDLib.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 axGrid1_ItemOleEvent(object sender,
AxEXGRIDLib._IGridEvents_ItemOleEventEvent e)
{
   System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
   for ( int i= 0; i < e.ev.CountParam ; i++ )
   {
      EXGRIDLib.IOleEventParam evP = e.ev[i];
      System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
```

```
evP.Value.ToString() );
    }
}
```

The following C# sample displays the event and its parameters when an user editor object fires an event:

```
private void axGrid1_UserEditorOleEvent(object sender,
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine("Event's name: " + e.ev.Name.ToString());
    for (int i = 0; i < e.ev.CountParam; i++)
    {
        EXGRIDLib.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 (ItemOleEvent event):

```
*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
        s = s +  "Name: " + ev.Param(i).Name + " ,Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s
```

The following VFP sample displays the event and its parameters when an user editor object fires an event (UserEditorOleEvent event):

```
*** ActiveX Control Event ***
LPARAMETERS object, ev, closeeditor, item, colindex

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 idenfier 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. For instance, tThe following VB sample closes the editor and focus a new column when user presses the TAB key inside an User editor:

```
Private Sub Grid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Item As EXGRIDLibCtl.HITEM, ByVal
ColIndex As Long)
  If (Ev.ID = -602) Then  ' KeyDown
    Dim iKey As Long
    iKey = Ev(0).Value
    If iKey = vbKeyTab Then
      With Grid1
        CloseEditor = True
        .FocusColumnIndex = .FocusColumnIndex + 1
        .SearchColumnIndex = .FocusColumnIndex
      End With
    End If
  End If
```

property OleEvent.Name as String

Retrieves the original name of the fired event.

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

Use the Name property to get the name of the event. Use the ID property to specify a specified even by its identifier. Use the ToString property to display information about fired event such us name, parameters, types and values. Use the CountParam property to count the parameters of an OLE event. Use the Param property to get the event's parameter. Use the Value property to specify the value of the parameter. The following VB sample enumerates the arguments of an OLE event when ItemOLEEvent or UserEditorOleEvent event is fired.

```
Private Sub Grid1_ItemOleEvent(ByVal Item As EXGRIDLibCtl.HITEM, ByVal Ev As
EXGRIDLibCtl.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 VB sample displays information about the fired event when the UserEditorOleEvent event occurs:

```
Private Sub Grid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Item As EXGRIDLibCtl.HITEM, ByVal
ColIndex As Long)
   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 <exgrid.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 OnItemOleEventGrid1(long Item, LPDISPATCH Ev)
{
    EXGRIDLib::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++ )
        {
```

```
        EXGRIDLib::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 EXGRIDLib namespace that include all objects and types of the control's TypeLibrary. In case your exgrid.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 C++ sample displays the event and its parameters when an user editor object fires an event:

```
void OnUserEditorOleEventGrid1(LPDISPATCH Object, LPDISPATCH Ev, BOOL FAR*
CloseEditor, long Item, long ColIndex)
{
    EXGRIDLib::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++ )
      {
        EXGRIDLib::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 following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
Private Sub AxGrid1_ItemOleEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_ItemOleEventEvent) Handles AxGrid1.ItemOleEvent
   Debug.WriteLine("Event's name: " & e.ev.Name)
   Dim i As Long
   For i = 0 To e.ev.CountParam - 1
      Dim eP As EXGRIDLib.OleEventParam
      eP = e.ev(i)
      Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
   Next
End Sub
```

The following VB.NET sample displays the event and its parameters when an user editor object fires an event:

```
Private Sub AxGrid1_UserEditorOleEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent) Handles AxGrid1.UserEditorOleEvent
   Debug.WriteLine("Event's name: " & e.ev.Name)
   Dim i As Long
   For i = 0 To e.ev.CountParam - 1
      Dim eP As EXGRIDLib.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 axGrid1_ItemOleEvent(object sender,
AxEXGRIDLib._IGridEvents_ItemOleEventEvent e)
{
   System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
   for ( int i= 0; i < e.ev.CountParam ; i++ )
   {
```

```
        EXGRIDLib.IOleEventParam evP = e.ev[i];
        System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
evP.Value.ToString() );
    }
}
```

The following C# sample displays the event and its parameters when an user editor object fires an event:

```
private void axGrid1_UserEditorOleEvent(object sender,
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine("Event's name: " + e.ev.Name.ToString());
    for (int i = 0; i < e.ev.CountParam; i++)
    {
        EXGRIDLib.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 (ItemOleEvent event):

```
*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
        s = s +  "Name: " + ev.Param(i).Name + " ,Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s
```

The following VFP sample displays the event and its parameters when an user editor object fires an event (UserEditorOleEvent event):

```
*** ActiveX Control Event ***
LPARAMETERS object, ev, closeeditor, item, colindex
```

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

Use the CountParam property to count the parameters of an OLE event. Use the Name property to get the parameter name. Use the Param property to get the event's parameter. Use the Value property to specify the value of the parameter. The following VB sample enumerates the arguments of an OLE event when ItemOLEEvent or UserEditorOleEvent event is fired.

```
Private Sub Grid1_ItemOleEvent(ByVal Item As EXGRIDLibCtl.HITEM, ByVal Ev As
EXGRIDLibCtl.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 VB sample displays information about the fired event when the UserEditorOleEvent event occurs:

```
Private Sub Grid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Item As EXGRIDLibCtl.HITEM, ByVal
ColIndex As Long)
   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 <exgrid.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 OnItemOleEventGrid1(long Item, LPDISPATCH Ev)
{
    EXGRIDLib::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++ )
        {
            EXGRIDLib::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 EXGRIDLib namespace that include all objects and types of the control's TypeLibrary. In case your exgrid.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 C++ sample displays the event and its parameters when an user editor object fires an event:

```
void OnUserEditorOleEventGrid1(LPDISPATCH Object, LPDISPATCH Ev, BOOL FAR*
CloseEditor, long Item, long ColIndex)
{
    EXGRIDLib::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++ )
        {
            EXGRIDLib::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 following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
Private Sub AxGrid1_ItemOleEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_ItemOleEventEvent) Handles AxGrid1.ItemOleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
    Dim i As Long
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXGRIDLib.OleEventParam
        eP = e.ev(i)
        Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
    Next
End Sub
```

The following VB.NET sample displays the event and its parameters when an user editor object fires an event:

```
Private Sub AxGrid1_UserEditorOleEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent) Handles AxGrid1.UserEditorOleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
    Dim i As Long
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXGRIDLib.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 axGrid1_ItemOleEvent(object sender,
AxEXGRIDLib._IGridEvents_ItemOleEventEvent e)
{
```

```
    System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
    for ( int i= 0; i < e.ev.CountParam ; i++ )
    {
        EXGRIDLib.IOleEventParam evP = e.ev[i];
        System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
evP.Value.ToString() );
    }
}
```

The following C# sample displays the event and its parameters when an user editor object fires an event:

```
private void axGrid1_UserEditorOleEvent(object sender,
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine("Event's name: " + e.ev.Name.ToString());
    for (int i = 0; i < e.ev.CountParam; i++)
    {
        EXGRIDLib.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 (ItemOleEvent event):

```
*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
        s = s +  "Name: " + ev.Param(i).Name + " ,Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s
```

The following VFP sample displays the event and its parameters when an user editor object fires an event (UserEditorOleEvent event):

```
*** ActiveX Control Event ***
LPARAMETERS object, ev, closeeditor, item, colindex

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 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.

Use the CountParam property to count the parameters of an OLE event. Use the Name property to get the parameter name. Use the Param property to get the event's parameter. Use the Value property to specify the value of the parameter. The following VB sample enumerates the arguments of an OLE event when ItemOLEEvent or UserEditorOleEvent event is fired.

```
Private Sub Grid1_ItemOleEvent(ByVal Item As EXGRIDLibCtl.HITEM, ByVal Ev As
EXGRIDLibCtl.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 VB sample displays information about the fired event when the UserEditorOleEvent event occurs:

```
Private Sub Grid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Item As EXGRIDLibCtl.HITEM, ByVal
ColIndex As Long)
   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 <exgrid.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 OnItemOleEventGrid1(long Item, LPDISPATCH Ev)
{
    EXGRIDLib::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++ )
        {
            EXGRIDLib::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 EXGRIDLib namespace that include all objects and types of the control's TypeLibrary. In case your exgrid.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 C++ sample displays the event and its parameters when an user editor object fires an event:

```
void OnUserEditorOleEventGrid1(LPDISPATCH Object, LPDISPATCH Ev, BOOL FAR*
CloseEditor, long Item, long ColIndex)
{
    EXGRIDLib::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++ )
      {
        EXGRIDLib::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 following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
Private Sub AxGrid1_ItemOleEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_ItemOleEventEvent) Handles AxGrid1.ItemOleEvent
   Debug.WriteLine("Event's name: " & e.ev.Name)
   Dim i As Long
   For i = 0 To e.ev.CountParam - 1
      Dim eP As EXGRIDLib.OleEventParam
      eP = e.ev(i)
      Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
   Next
End Sub
```

The following VB.NET sample displays the event and its parameters when an user editor object fires an event:

```
Private Sub AxGrid1_UserEditorOleEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent) Handles AxGrid1.UserEditorOleEvent
   Debug.WriteLine("Event's name: " & e.ev.Name)
   Dim i As Long
   For i = 0 To e.ev.CountParam - 1
      Dim eP As EXGRIDLib.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 axGrid1_ItemOleEvent(object sender,
AxEXGRIDLib._IGridEvents_ItemOleEventEvent e)
{
   System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
   for ( int i= 0; i < e.ev.CountParam ; i++ )
   {
      EXGRIDLib.IOleEventParam evP = e.ev[i];
      System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
```

```
evP.Value.ToString() );
    }
}
```

The following C# sample displays the event and its parameters when an user editor object fires an event:

```
private void axGrid1_UserEditorOleEvent(object sender,
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine("Event's name: " + e.ev.Name.ToString());
    for (int i = 0; i < e.ev.CountParam; i++)
    {
        EXGRIDLib.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 (ItemOleEvent event):

```
*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
        s = s +  "Name: " + ev.Param(i).Name + " ,Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s
```

The following VFP sample displays the event and its parameters when an user editor object fires an event (UserEditorOleEvent event):

```
*** ActiveX Control Event ***
LPARAMETERS object, ev, closeeditor, item, colindex

local s
```

```
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
	s = s +  "Name: " + ev.Param(i).Name + " ,Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s
```

property OleEventParam.Value as Variant

Specifies the value of the event's parameter.

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

Use the CountParam property to count the parameters of an OLE event. Use the Name property to get the parameter name. Use the Param property to get the event's parameter. Use the Value property to specify the value of the parameter. The following VB sample enumerates the arguments of an OLE event when ItemOLEEvent or UserEditorOleEvent event is fired.

```
Private Sub Grid1_ItemOleEvent(ByVal Item As EXGRIDLibCtl.HITEM, ByVal Ev As
EXGRIDLibCtl.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 VB sample displays information about the fired event when the UserEditorOleEvent event occurs:

```
Private Sub Grid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Item As EXGRIDLibCtl.HITEM, ByVal
ColIndex As Long)
   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 <exgrid.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 OnItemOleEventGrid1(long Item, LPDISPATCH Ev)
{
    EXGRIDLib::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++ )
        {
            EXGRIDLib::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 EXGRIDLib namespace that include all objects and types of the control's TypeLibrary. In case your exgrid.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 C++ sample displays the event and its parameters when an user editor object fires an event:

```
void OnUserEditorOleEventGrid1(LPDISPATCH Object, LPDISPATCH Ev, BOOL FAR*
CloseEditor, long Item, long ColIndex)
{
    EXGRIDLib::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++ )
      {
        EXGRIDLib::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 following VB.NET sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
Private Sub AxGrid1_ItemOleEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_ItemOleEventEvent) Handles AxGrid1.ItemOleEvent
   Debug.WriteLine("Event's name: " & e.ev.Name)
   Dim i As Long
   For i = 0 To e.ev.CountParam - 1
      Dim eP As EXGRIDLib.OleEventParam
      eP = e.ev(i)
      Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
   Next
End Sub
```

The following VB.NET sample displays the event and its parameters when an user editor object fires an event:

```
Private Sub AxGrid1_UserEditorOleEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent) Handles AxGrid1.UserEditorOleEvent
   Debug.WriteLine("Event's name: " & e.ev.Name)
   Dim i As Long
   For i = 0 To e.ev.CountParam - 1
      Dim eP As EXGRIDLib.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 axGrid1_ItemOleEvent(object sender,
AxEXGRIDLib._IGridEvents_ItemOleEventEvent e)
{
   System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
   for ( int i= 0; i < e.ev.CountParam ; i++ )
   {
      EXGRIDLib.IOleEventParam evP = e.ev[i];
      System.Diagnostics.Debug.WriteLine( "Name: " + evP.Name.ToString() + ", Value: " +
```

```
evP.Value.ToString() );
    }
}
```

The following C# sample displays the event and its parameters when an user editor object fires an event:

```
private void axGrid1_UserEditorOleEvent(object sender,
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine("Event's name: " + e.ev.Name.ToString());
    for (int i = 0; i < e.ev.CountParam; i++)
    {
        EXGRIDLib.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 (ItemOleEvent event):

```
*** ActiveX Control Event ***
LPARAMETERS item, ev

local s
s = "Event's name: " + ev.Name
for i = 0 to ev.CountParam - 1
        s = s +  "Name: " + ev.Param(i).Name + " ,Value: " + Str(ev.Param(i).Value)
endfor
wait window nowait s
```

The following VFP sample displays the event and its parameters when an user editor object fires an event (UserEditorOleEvent event):

```
*** ActiveX Control Event ***
LPARAMETERS object, ev, closeeditor, item, colindex

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
```

UnboundHandler object

The control supports unbound mode. In unbound mode, the user is responsible for retrieving items. The unbound mode and virtual unbound modes were provided to let user displays large number of items. In order to let the control works in unbound mode, the user has to implement the IUnboundHandler notification interface. Use the VirtualMode property to run the control in virtual mode. The UnboundHandler property specifies the control's unbound handler. Currently, the UnboundHandler / IUnboundHandler interface is available for /COM version only.

Here's the IDL definition of the IUnboundHandler interface:

```
[
   uuid(BA3AA5FA-5B09-40F6-80DF-B051C20150B6),
   pointer_default(unique)
]
interface IUnboundHandler : IUnknown
{
   [propget, id(1), helpcontext(3001), helpstring("Gets the number of items.")] HRESULT
ItemsCount( IDispatch* Source, [out, retval ] long* pVal );
   [id(2), helpcontext(3002), helpstring("The source requires an item.")] HRESULT
ReadItem( long Index, IDispatch* Source, long ItemHandle );
}
```

Here's the IDL definition of the UnboundHandler interface (this interface is available starting from the version 11.1):

```
[
   uuid(BA3AA5FA-5B09-40F6-80DF-B051C20150B7),
]
dispinterface UnboundHandler
{
   interface IUnboundHandler;
}
```

The following **VB** sampledisplays 1,000,000 items in virtual mode:

```
Implements EXGRIDLibCtl.IUnboundHandler

Private Sub Form_Load()
```

```
    With Grid1
        .BeginUpdate
        .Columns.Add("Index").FormatColumn = "value format `0`"
        .VirtualMode = True
        Set .UnboundHandler = Me
        .EndUpdate
    End With
End Sub

Private Property Get IUnboundHandler_ItemsCount(ByVal Source As Object) As Long
    IUnboundHandler_ItemsCount = 1000000
End Property

Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As Object,
ByVal ItemHandle As Long)
    With Source.Items
        .CellValue(ItemHandle, 0) = Index + 1
    End With
End Sub
```

The following **VB/NET** sampledisplays 1,000,000 items in virtual mode:

```
Public Class Form1
    Implements EXGRIDLib.IUnboundHandler

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        With AxGrid1
            .BeginUpdate()
            .Columns.Add("Index").FormatColumn = "value format `0`"
            .VirtualMode = True
            .UnboundHandler = Me
            .EndUpdate()
        End With
    End Sub

    Public ReadOnly Property ItemsCount(ByVal Source As Object) As Integer Implements
```

```
EXGRIDLib.IUnboundHandler.ItemsCount
        Get
            ItemsCount = 10000000
        End Get
    End Property

    Public Sub ReadItem(ByVal Index As Integer, ByVal Source As Object, ByVal ItemHandle
As Integer) Implements EXGRIDLib.IUnboundHandler.ReadItem
        With Source.Items
            .CellValue(ItemHandle, 0) = Index + 1
        End With
    End Sub
End Class
```

The following **C#** sampledisplays 1,000,000 items in virtual mode:

```
public partial class Form1 : Form, EXGRIDLib.IUnboundHandler
{
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        axGrid1.BeginUpdate();
        (axGrid1.Columns.Add("Index") as EXGRIDLib.IColumn).FormatColumn = "value
format `0`";
        axGrid1.VirtualMode = true;
        axGrid1.UnboundHandler = this;
        axGrid1.EndUpdate();
    }

    public int get_ItemsCount(object Source)
    {
        return 1000000;
    }
```

```
    public void ReadItem(int Index, object Source, int ItemHandle)
    {
        (Source as EXGRIDLib.IGrid).Items.set_CellValue(ItemHandle, 0, Index + 1);
    }
}
```

The following **VFP 9** sample displays 1,000,000 items in virtual mode:

```
with thisform.Grid1
    .Columns.Add("Index").FormatColumn = "value format `0`"
    .VirtualMode = .T.
    .UnboundHandler = newobject('UnboundHandler', 'class1.prg')
endwith
```

where the class1.prg is:

```
define class UnboundHandler as session OLEPUBLIC

implements IUnboundHandler in "ExGrid.dll"
function IUnboundHandler_get_ItemsCount(Source)
    return 1000000
endfunc
function IUnboundHandler_ReadItem(Index, Source, ItemHandle)
    With Source.Items
        .CellValue(ItemHandle, 0) = Index + 1
    EndWith
endfunc

implements UnboundHandler in "ExGrid.dll"
function UnboundHandler_get_ItemsCount(Source)
    return this.IUnboundHandler_get_ItemsCount(Source)
endfunc
function UnboundHandler_ReadItem(Index, Source, ItemHandle)
    return this.IUnboundHandler_ReadItem(Index, Source, ItemHandle)
endfunc

enddefine
```

The following **VFP 7** and **VFP 8** sample displays 1,000,000 items in virtual mode:

```
with thisform.Grid1
   .Columns.Add("Index").FormatColumn = "value format `0`"
   .VirtualMode = .T.
   .UnboundHandler = newobject('UnboundHandler', 'class1.prg')
endwith
```

where the class1.prg is:

```
define class UnboundHandler as custom

implements IUnboundHandler in "ExGrid.dll"

function IUnboundHandler_get_ItemsCount(Source)
   return 10000000
endfunc

function IUnboundHandler_ReadItem(Index, Source, ItemHandle)
   With Source.Items
      .DefaultItem = ItemHandle
      .CellValue(0, 0) = Index + 1
   EndWith
endfunc

enddefine
```

The UnboundHandler / IUnboundHandler interface requires the following properties and methods:

Name	Description
ItemsCount	Gets the number of items.
ReadItem	The source requires an item.

property UnboundHandler.ItemsCount (Source as Object) as Long

Gets the number of items.

Type	Description
Source as Object	The control that requires the number of items
Long	A Long expression that specifies the number of items in unbound/virtual mode.

The ItemsCount property specifies the number of items in unbound/virtual mode.

The following **VB** sample, shows how ItemsCount property should be implemented:

```
Private Property Get IUnboundHandler_ItemsCount(ByVal Source As Object) As Long
   IUnboundHandler_ItemsCount = 1000000
End Property
```

The following **VB/NET** sample, shows how ItemsCount property should be implemented:

```
Public ReadOnly Property ItemsCount(ByVal Source As Object) As Integer Implements
EXGRIDLib.IUnboundHandler.ItemsCount
   Get
      ItemsCount = 10000000
   End Get
End Property
```

The following **C#** sample, shows how ItemsCount property should be implemented:

```
public int get_ItemsCount(object Source)
{
   return 1000000;
}
```

The following **VFP** sample , shows how ItemsCount property should be implemented:

```
function IUnboundHandler_get_ItemsCount(Source)
   return 1000000
endfunc
```

method UnboundHandler.ReadItem (Index as Long, Source as Object, ItemHandle as Long)

The source requires an item.

Type	Description
Index as Long	A Long expression that specifies the index of the item to be requested
Source as Object	The source object to be filled.
ItemHandle as Long	A Long expression that specifies the handle of the item in the source, that's associated with the requested index.

The ReadItem method is called every time the Source requires an item to be displayed.

The following **VB** sample, shows how ReadItem method should be implemented:

```
Private Sub IUnboundHandler_ReadItem(ByVal Index As Long, ByVal Source As Object,
ByVal ItemHandle As Long)
   With Source.Items
      .CellValue(ItemHandle, 0) = Index + 1
   End With
End Sub
```

The following **VB/NET** sample, shows how ReadItem method should be implemented:

```
Public Sub ReadItem(ByVal Index As Integer, ByVal Source As Object, ByVal ItemHandle As
Integer) Implements EXGRIDLib.IUnboundHandler.ReadItem
   With Source.Items
      .CellValue(ItemHandle, 0) = Index + 1
   End With
End Sub
```

The following **C#** sample, shows how ReadItem method should be implemented:

```
public void ReadItem(int Index, object Source, int ItemHandle)
{
   (Source as EXGRIDLib.IGrid).Items.set_CellValue(ItemHandle, 0, Index + 1);
}
```

The following **VFP** sample , shows how ReadItem method should be implemented:

```
function IUnboundHandler_ReadItem(Index, Source, ItemHandle)
    With Source.Items
        .CellValue(ItemHandle, 0) = Index + 1
    EndWith
endfunc
```

ExGrid events

The exGrid component supports the following events:

Name	Description
AddColumn	Fired after a new column has been added.
AddGroupItem	Occurs after a new Group Item has been inserted to Items collection.
AddItem	Occurs after a new Item has been inserted to Items collection.
AfterExpandItem	Fired after an item is expanded (collapsed).
AnchorClick	Occurs when an anchor element is clicked.
BeforeExpandItem	Fired before an item is about to be expanded (collapsed).
ButtonClick	Occurs when user clicks on the cell's button.
CellImageClick	Fired after the user clicks on the image's cell area.
CellStateChanged	Fired after cell's state has been changed.
CellStateChanging	Fired before cell's state is about to be changed.
Change	Occurs when the user changes the cell's content.
Click	Occurs when the user presses and then releases the left mouse button over the grid control.
ColumnClick	Fired after the user clicks on column's header.
DblClick	Occurs when the user dblclk the left mouse button over an object.
Edit	Occurs just before editing the focused cell.
EditClose	Occurs when the edit operation ends.
EditOpen	Occurs when the edit operation starts.
Error	Fired when an internal error occurs.
Event	Notifies the application once the control fires an event.
FilterChange	Occurs when filter was changed.
FilterChanging	Notifies your application that the filter is about to change.
FocusChanged	Occurs when a new cell is focused.
FormatColumn	Fired when a cell requires to format its value.
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.

URChange	Occurs once the control's undo/redo queue is changed.
UserEditorClose	Fired the user editor is about to be opened.
UserEditorOleEvent	Occurs when an user editor fires an event.
UserEditorOpen	Occurs when an user editor is about to be opened.
ValidateValue	Occurs before user changes the cell's value.

event AddColumn (Column as Column)

Fired after a new column has been added.

Type	Description
Column as Column	A Column object being inserted to the Columns collection.

Use the AddColumn event to notify your application that a new column has been added. Use Add method to add new columns to the control. The AddColumn event is called even if the user binds the control to an ADO recordset using DataSource property. Use the Def property to specify default values for certain properties of a Column object. Use the AddColumn property to associate extra data to columns being added. The ColumnAutoResize property specifies whether the columns fit the control's client area. Use the Width property to specify the column's width.

Syntax for AddColumn event, **/NET** version, on:

C#

```
private void AddColumn(object sender,exontrol.EXGRIDLib.Column Column)
{
}
```

VB

```
Private Sub AddColumn(ByVal sender As System.Object,ByVal Column As
exontrol.EXGRIDLib.Column) Handles AddColumn
End Sub
```

Syntax for AddColumn event, **/COM** version, on:

C#

```
private void AddColumn(object sender,
AxEXGRIDLib._IGridEvents_AddColumnEvent e)
{
}
```

C++

```
void OnAddColumn(LPDISPATCH Column)
{
}
```

C++ Builder

```
void __fastcall AddColumn(TObject *Sender,Exgridlib_tlb::IColumn *Column)
{
}
```

Delphi

```
procedure AddColumn(ASender: TObject; Column : IColumn);
begin
end;
```

Delphi 8 (.NET only)

```
procedure AddColumn(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_AddColumnEvent) Handles AddColumn
End Sub
```

VB6

```
Private Sub AddColumn(ByVal Column As EXGRIDLibCtl.IColumn)
End Sub
```

VBA

```
Private Sub AddColumn(ByVal Column As Object)
End Sub
```

VFP

```
LPARAMETERS Column
```

Xbas…

```
PROCEDURE OnAddColumn(oGrid,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.Grid.1::IColumn)
end function
```

dBASE

```
function nativeObject_AddColumn(Column)
return
```

The following VB sample changes the column's width, when adding the column:

```
Private Sub Grid1_AddColumn(ByVal Column As EXGRIDLibCtl.IColumn)
    Column.Width = 128
End Sub
```

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

```
#include "Column.h"
void OnAddColumnGrid1(LPDISPATCH Column)
{
    CColumn column( Column );column.m_bAutoRelease = FALSE;
    column.SetWidth( 128 );
}
```

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

```
Private Sub AxGrid1_AddColumn(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_AddColumnEvent) Handles AxGrid1.AddColumn
    e.column.Width = 128
```

```
End Sub
```

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

```
private void axGrid1_AddColumn(object sender,
AxEXGRIDLib._IGridEvents_AddColumnEvent e)
{
    e.column.Width = 128;
}
```

The following VFP sample changes the column's width:

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

with column
    .Width = 128
endwith
```

event AddGroupItem (Item as HITEM)

Occurs after a new Group Item has been inserted to Items collection.

Type	Description
Item as HITEM	A Long expression that indicates the handle of the grouping items being inserted.

The AddGroupItem event is fired for each new item to be inserted in the Items collection during the grouping. The GroupItem method determines the index of the column that indicates the column being grouped. In other words, the CellCaption(Item,GroupItem(Item)) gets the default caption to be displayed for the grouping item. The Ungroup method removes all grouping items. For instance, when a column gets grouped by, the control sorts by that column, collects the unique values being found, and add a new item for each value found, by adding the items of the same value as children.

The AddGroupItem event can be used in any of the followings:

- customize the visual appearance for any grouping item,
- customize the aggregate formula to be displayed instead, MAX, MIN, COUNT, and so on,
- adding new headers or footers for grouping items

Syntax for AddGroupItem event, **/NET** version, on:

C#
```
private void AddGroupItem(object sender,int Item)
{
}
```

VB
```
Private Sub AddGroupItem(ByVal sender As System.Object,ByVal Item As Integer)
Handles AddGroupItem
End Sub
```

Syntax for AddGroupItem event, **/COM** version, on:

C#
```
private void AddGroupItem(object sender,
AxEXGRIDLib._IGridEvents_AddGroupItemEvent e)
{
}
```

C++
```
void OnAddGroupItem(long Item)
{
```

```
}
```

C++ Builder

```
void __fastcall AddGroupItem(TObject *Sender,Exgridlib_tlb::HITEM Item)
{
}
```

Delphi

```
procedure AddGroupItem(ASender: TObject; Item : HITEM);
begin
end;
```

Delphi 8 (.NET only)

```
procedure AddGroupItem(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_AddGroupItemEvent);
begin
end;
```

Powe…

```
begin event AddGroupItem(long Item)
end event AddGroupItem
```

VB.NET

```
Private Sub AddGroupItem(ByVal sender As System.Object, ByVal e As
AxEXGRIDLib._IGridEvents_AddGroupItemEvent) Handles AddGroupItem
End Sub
```

VB6

```
Private Sub AddGroupItem(ByVal Item As EXGRIDLibCtl.HITEM)
End Sub
```

VBA

```
Private Sub AddGroupItem(ByVal Item As Long)
End Sub
```

VFP

```
LPARAMETERS Item
```

Xbas…

```
PROCEDURE OnAddGroupItem(oGrid,Item)
RETURN
```

Syntax for AddGroupItem event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="AddGroupItem(Item)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function AddGroupItem(Item)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComAddGroupItem HITEM llItem
   Forward Send OnComAddGroupItem llItem
End_Procedure
```

Visual Objects

```
METHOD OCX_AddGroupItem(Item) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AddGroupItem(int _Item)
{
}
```

XBasic

```
function AddGroupItem as v (Item as OLE::Exontrol.Grid.1::HITEM)
end function
```

dBASE

```
function nativeObject_AddGroupItem(Item)
return
```

The following samples shows how to underline the grouping items, and add a footer item to show the total/sum on the column with the index 6:

VBA

```
Private Sub Grid1_AddGroupItem(ByVal Item As Long)
   With Grid1
      With .Items
         .CellUnderline(Item,.GroupItem(Item)) = True
         h = .InsertItem(Item,0,"")
         .SelectableItem(h) = False
         .CellValue(h,6) = "sum(parent,rec,dbl(%6))"
         .CellValueFormat(h,6) = 5 ' ValueFormatEnum.exTotalField Or
ValueFormatEnum.exHTML
         .FormatCell(h,6) = "`<font ;7><b>Sum</b>: ` + value"
      End With
```

```
    End With
End Sub
```

VB

```
Private Sub Grid1_AddGroupItem(ByVal Item As EXGRIDLibCtl.HITEM)
    With Grid1
        With .Items
            .CellUnderline(Item,.GroupItem(Item)) = True
            h = .InsertItem(Item,0,"")
            .SelectableItem(h) = False
            .CellValue(h,6) = "sum(parent,rec,dbl(%6))"
            .CellValueFormat(h,6) = ValueFormatEnum.exTotalField Or
ValueFormatEnum.exHTML
            .FormatCell(h,6) = "`<font ;7><b>Sum</b>:` + value"
        End With
    End With
End Sub
```

VB.NET

```
Private Sub Exgrid1_AddGroupItem(ByVal sender As System.Object,ByVal Item As Integer)
Handles Exgrid1.AddGroupItem
    Dim h
    With Exgrid1
        With .Items
            .set_CellUnderline(Item,.get_GroupItem(Item),True)
            h = .InsertItem(Item,0,"")
            .set_SelectableItem(h,False)
            .set_CellValue(h,6,"sum(parent,rec,dbl(%6))")
            .set_CellValueFormat(h,6,exontrol.EXGRIDLib.ValueFormatEnum.exTotalField Or
exontrol.EXGRIDLib.ValueFormatEnum.exHTML)
            .set_FormatCell(h,6,"`<font ;7><b>Sum</b>:` + value")
        End With
    End With
End Sub
```

C++

```
void OnAddGroupItemGrid1(long Item)
{
   /*
      Copy and paste the following directives to your header file as
      it defines the namespace 'EXGRIDLib' for the library: 'ExGrid 1.0 Control Library'
      #import <ExGrid.dll>
      using namespace EXGRIDLib;
   */
   EXGRIDLib::IGridPtr spGrid1 = GetDlgItem(IDC_GRID1)->GetControlUnknown();
   EXGRIDLib::IItemsPtr var_Items = spGrid1->GetItems();
      var_Items->PutCellUnderline(Item,var_Items->GetGroupItem(Item),VARIANT_TRUE);
      long h = var_Items->InsertItem(Item,long(0),"");
      var_Items->PutSelectableItem(h,VARIANT_FALSE);
      var_Items->PutCellValue(h,long(6),"sum(parent,rec,dbl(%6))");
      var_Items-
>PutCellValueFormat(h,long(6),EXGRIDLib::ValueFormatEnum(EXGRIDLib::exTotalField |
EXGRIDLib::exHTML));
      var_Items->PutFormatCell(h,long(6),L"`<font ;7><b>Sum</b>:` + value");
}
```

C++ Builder

```
void __fastcall TForm1::Grid1AddGroupItem(TObject *Sender,Exgridlib_tlb::HITEM Item)
{
   Exgridlib_tlb::IItemsPtr var_Items = Grid1->Items;
      var_Items->set_CellUnderline(TVariant(Item),TVariant(var_Items-
>get_GroupItem(Item)),true);
      long h = var_Items->InsertItem(Item,TVariant(0),TVariant(""));
      var_Items->set_SelectableItem(h,false);
      var_Items->set_CellValue(TVariant(h),TVariant(6),TVariant("sum(parent,rec,dbl(%6))"));
      var_Items-
>set_CellValueFormat(TVariant(h),TVariant(6),Exgridlib_tlb::ValueFormatEnum::exTotalField
| Exgridlib_tlb::ValueFormatEnum::exHTML);
      var_Items->set_FormatCell(TVariant(h),TVariant(6),L"`<font ;7><b>Sum</b>:` +
value");
}
```

C#

```
private void exgrid1_AddGroupItem(object sender,int Item)
{
   exontrol.EXGRIDLib.Items var_Items = exgrid1.Items;
      var_Items.set_CellUnderline(Item,var_Items.get_GroupItem(Item),true);
      int h = var_Items.InsertItem(Item,0,"");
      var_Items.set_SelectableItem(h,false);
      var_Items.set_CellValue(h,6,"sum(parent,rec,dbl(%6))");

var_Items.set_CellValueFormat(h,6,exontrol.EXGRIDLib.ValueFormatEnum.exTotalField |
exontrol.EXGRIDLib.ValueFormatEnum.exHTML);
      var_Items.set_FormatCell(h,6,"`<font ;7><b>Sum</b>:` + value");
}
```

JavaScript

```
<SCRIPT FOR="Grid1" EVENT="AddGroupItem(Item)" LANGUAGE="JScript">
   var var_Items = Grid1.Items;
      var_Items.CellUnderline(Item,var_Items.GroupItem(Item)) = true;
      var h = var_Items.InsertItem(Item,0,"");
      var_Items.SelectableItem(h) = false;
      var_Items.CellValue(h,6) = "sum(parent,rec,dbl(%6))";
      var_Items.CellValueFormat(h,6) = 5;
      var_Items.FormatCell(h,6) = "`<font ;7><b>Sum</b>:` + value";
</SCRIPT>
```

VFP

```
*** AddGroupItem event - Occurs after a new Group Item has been inserted to Items
collection. ***
LPARAMETERS Item
   with thisform.Grid1
      with .Items
         .CellUnderline(Item,.GroupItem(Item)) = .T.
         h = .InsertItem(Item,0,"")
         .SelectableItem(h) = .F.
         .CellValue(h,6) = "sum(parent,rec,dbl(%6))"
         .CellValueFormat(h,6) = 5 && ValueFormatEnum.exTotalField Or
ValueFormatEnum.exHTML
```

```
        .FormatCell(h,6) = "`<font ;7><b>Sum</b>:` + value"
      endwith
   endwith
```

Delphi

```
procedure TForm1.Grid1AddGroupItem(ASender: TObject; Item : HITEM);
begin
   with Grid1 do
   begin
      with Items do
      begin
         CellUnderline[OleVariant(Item),OleVariant(GroupItem[Item])] := True;
         h := InsertItem(Item,OleVariant(0),'');
         SelectableItem[h] := False;
         CellValue[OleVariant(h),OleVariant(6)] := 'sum(parent,rec,dbl(%6))';
         CellValueFormat[OleVariant(h),OleVariant(6)] :=
Integer(EXGRIDLib_TLB.exTotalField) Or Integer(EXGRIDLib_TLB.exHTML);
         FormatCell[OleVariant(h),OleVariant(6)] := '`<font ;7><b>Sum</b>:` + value';
      end;
   end
end;
```

Visual Objects

```
METHOD OCX_Exontrol1AddGroupItem(Item) CLASS MainDialog
   local var_Items as IItems
   local h as USUAL
   var_Items := oDCOCX_Exontrol1:Items
      var_Items:[CellUnderline,Item,var_Items:[GroupItem,Item]] := true
      h := var_Items:InsertItem(Item,0,"")
      var_Items:[SelectableItem,h] := false
      var_Items:[CellValue,h,6] := "sum(parent,rec,dbl(%6))"
      var_Items:[CellValueFormat,h,6] := exTotalField | exHTML
      var_Items:[FormatCell,h,6] := "`<font ;7><b>Sum</b>:` + value"
RETURN NIL
```

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 newly inserted item.

Use the AddItem event to notify your application that a new item has been inserted into the Items collection. The AddItem, InsertItem and InsertControlItem methods fire the AddItem event. The PutItems method invokes the AddItem event each time a new item is added. If the user binds the control to an ADO recordset using the DataSource property, AddEvent is called each time the control inserts a new item.

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, AxEXGRIDLib._IGridEvents_AddItemEvent e)
{
}
```

C++
```
void OnAddItem(long Item)
{
}
```

C++ Builder
```
void __fastcall AddItem(TObject *Sender,Exgridlib_tlb::HITEM Item)
{
}
```

Delphi
```
procedure AddItem(ASender: TObject; Item : HITEM);
begin
```

```
end;
```

Delphi 8 (.NET only)

```
procedure AddItem(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_AddItemEvent) Handles AddItem
End Sub
```

VB6

```
Private Sub AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
End Sub
```

VBA

```
Private Sub AddItem(ByVal Item As Long)
End Sub
```

VFP

```
LPARAMETERS Item
```

Xbas…

```
PROCEDURE OnAddItem(oGrid,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.Grid.1::HITEM)
end function
```

dBASE

```
function nativeObject_AddItem(Item)
return
```

The following VB sample changes the item's background color:

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
   With Grid1.Items
      .ItemBackColor(Item) = IIf(.ItemToIndex(Item) Mod 2 = 0, vbBlue, vbRed)
   End With
End Sub
```

The following VB sample adds WS_HSCROLL and WS_VSCROLL window styles to the container window that hosts an ActiveX control

```
Private Const WS_VSCROLL = &H200000
Private Const WS_HSCROLL = &H100000
```

```
...
```

```
With Grid1.Items
   .InsertControlItem , "https://www.exontrol.com"
End With
```

```
...
```

```
Private Sub Grid1_AddItem(ByVal Item As EXGRIDLibCtl.HITEM)
```

```
With Grid1.Items
   If (.ItemControlID(Item) Like "http://www.*") Then
      ' Some of controls like the WEB control, require some additional window styles ( like
WS_HSCROLL and WS_VSCROLL window styles )
      ' for the window that hosts that WEB control, to allow scrolling the web page
      .ItemWindowHostCreateStyle(Item) = .ItemWindowHostCreateStyle(Item) +
WS_HSCROLL + WS_VSCROLL
   End If
   End With
End Sub
```

The following C++ sample changes the item's foreground color when a new items is inserted:

```
#include "Items.h"
void OnAddItemGrid1(long Item)
{
   if ( ::IsWindow( m_grid.m_hWnd ) )
   {
      CItems items = m_grid.GetItems();
      items.SetItemForeColor( Item, RGB(0,0,255) );
   }
}
```

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 AxGrid1_AddItem(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_AddItemEvent) Handles AxGrid1.AddItem
   AxGrid1.Items.ItemForeColor(e.item) = ToUInt32(Color.Blue)
```

```
End Sub
```

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 axGrid1_AddItem(object sender, AxEXGRIDLib._IGridEvents_AddItemEvent e)
{
   axGrid1.Items.set_ItemForeColor(e.item, ToUInt32(Color.Blue));
}
```

The following VFP sample changes the item's foreground color when a new items is inserted:

```
*** ActiveX Control Event ***
LPARAMETERS item

with thisform.Grid1.Items
   .DefaultItem = item
   .ItemForeColor( 0 ) = RGB(0,0,255 )
endwith
```

event AfterExpandItem (Item as HITEM)

Fired after an item is expanded (collapsed).

Type	Description
Item as HITEM	A long expression that specifies the handle of the item that is 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,
AxEXGRIDLib._IGridEvents_AfterExpandItemEvent e)
{
}
```

C++

```
void OnAfterExpandItem(long Item)
{
}
```

C++ Builder

```
void __fastcall AfterExpandItem(TObject *Sender,Exgridlib_tlb::HITEM Item)
{
}
```

Delphi

```
procedure AfterExpandItem(ASender: TObject; Item : HITEM);
```

```
begin
end;
```

Delphi 8 (.NET only)

```
procedure AfterExpandItem(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_AfterExpandItemEvent) Handles AfterExpandItem
End Sub
```

VB6

```
Private Sub AfterExpandItem(ByVal Item As EXGRIDLibCtl.HITEM)
End Sub
```

VBA

```
Private Sub AfterExpandItem(ByVal Item As Long)
End Sub
```

VFP

```
LPARAMETERS Item
```

Xbas…

```
PROCEDURE OnAfterExpandItem(oGrid,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.Grid.1::HITEM)
end function
```

dBASE

```
function nativeObject_AfterExpandItem(Item)
return
```

The following VB sample displays whether an item is expanded or collapsed:

```
Private Sub Grid1_AfterExpandItem(ByVal Item As EXGRIDLibCtl.HITEM)
   With Grid1.Items
      Debug.Print "Item is " & IIf(.ExpandItem(Item), "expanded", "collapsed") & "."
   End With
End Sub
```

The following C++ sample prints the item's state when it is expanded or collapsed:

```
#include "Items.h"
void OnAfterExpandItemGrid1(long Item)
{
   if ( ::IsWindow( m_grid.m_hWnd ) )
   {
      CItems items = m_grid.GetItems();
      CString strFormat;
      strFormat.Format( "%s", items.GetExpandItem( Item ) ? "expanded" : "collapsed" );
      OutputDebugString( strFormat );
```

```
    }
}
```

The following C# sample prints the item's state when it is expanded or collapsed:

```
private void axGrid1_AfterExpandItem(object sender,
AxEXGRIDLib._IGridEvents_AfterExpandItemEvent e)
{
    System.Diagnostics.Debug.WriteLine(axGrid1.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 AxGrid1_AddItem(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_AddItemEvent) Handles AxGrid1.AddItem
    AxGrid1.Items.ItemForeColor(e.item) = ToUInt32(Color.Blue)
End Sub
```

The following VFP sample prints the item's state when it is expanded or collapsed:

```
*** ActiveX Control Event ***
LPARAMETERS item

with thisform.Grid1.Items
    if ( .ExpandItem(item) )
        wait window "expanded" nowait
    else
        wait window "collapsed" nowait
    endif
endwith
```

event AnchorClick (AnchorID as String, Options as String)

Occurs when an anchor element is clicked.

Type	Description
AnchorID as String	A string expression that indicates the identifier of the anchor.
Options as String	A string expression that specifies options of the anchor element.

The control fires the AnchorClick event to notify that the user clicks an anchor element. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The **<a>** element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The AnchorClick event is fired only if prior clicking the control it shows the hand cursor. For instance, if the cell is disabled, the hand cursor is not shown when hovers the anchor element, and so the AnchorClick event is not fired. Use the FormatAnchor property to specify the visual effect for anchor elements. For instance, if the user clicks the anchor *<a1>anchor*, the control fires the AnchorClick event, where the AnchorID parameter is 1, and the Options parameter is empty. Also, if the user clicks the anchor *<a 1;yourextradata>anchor*, the AnchorID parameter of the AnchorClick event is 1, and the Options parameter is "yourextradata". Use the AnchorFromPoint property to retrieve the identifier of the anchor element from the cursor.

Syntax for AnchorClick event, **/NET** version, on:

C#

```
private void AnchorClick(object sender,string AnchorID,string Options)
{
}
```

VB

```
Private Sub AnchorClick(ByVal sender As System.Object,ByVal AnchorID As
String,ByVal Options As String) Handles AnchorClick
End Sub
```

Syntax for AnchorClick event, **/COM** version, on:

C#

```
private void AnchorClick(object sender,
AxEXGRIDLib._IGridEvents_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_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(oGrid,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 BeforeExpandItem (Item as HITEM, Cancel as Variant)

Fired before an item is about to be expanded (collapsed).

Type	Description
Item as HITEM	A long expression that indicates the item being expanded or collapsed.
Cancel as Variant	A boolean expression that indicates whether the expanding or collapsing operation is canceled.

Use the AfterExpandItem and BeforeExpandItem events to notify your application that an item is expanded or collapsed. Use the ExpandItem property to expand or collapse items at runtime. Use the BeforeExpandItem event to disable expanding or collapsing the items. Use the ItemHasChildren property to specify whether the control should display a + sign to the items, even they have no child items, so you can build a virtual tree. A virtual tree can load items as soon as user expands an item. Use the ExpandOnSearch property to expand items while user types characters to search for items using incremental search feature. Use the ChildCount property to get the number of child items. Use the ItemChild property to retrieve the first child item.

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,
AxEXGRIDLib._IGridEvents_BeforeExpandItemEvent e)
{
}
```

C++

```
void OnBeforeExpandItem(long Item,VARIANT FAR* Cancel)
{
}
```

C++ Builder

```
void __fastcall BeforeExpandItem(TObject *Sender,Exgridlib_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_BeforeExpandItemEvent) Handles BeforeExpandItem
End Sub
```

VB6

```
Private Sub BeforeExpandItem(ByVal Item As EXGRIDLibCtl.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(oGrid,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.Grid.1::HITEM,Cancel as A)
end function
```

dBASE

```
function nativeObject_BeforeExpandItem(Item,Cancel)
return
```

The following VB sample disables expanding or collapsing items:

```
Private Sub Grid1_BeforeExpandItem(ByVal Item As EXGRIDLibCtl.HITEM, Cancel As
Variant)
   Cancel = True
End Sub
```

Use the BeforeExpandItem event to add child items when an item is expanded, and has the ItemHasChildren property on True, like in the following VB sample:

```
Private Sub Grid1_BeforeExpandItem(ByVal Item As EXGRIDLibCtl.HITEM, Cancel As
Variant)
   With Grid1.Items
      If (.ItemHasChildren(Item)) Then
```

```
        If (.ChildCount(Item) = 0) Then
            .InsertItem Item, , "new item " & Item
        End If
    End If
  End With
End Sub
```

The following C++ sample cancels expanding or collapsing items:

```
void OnBeforeExpandItemGrid1(long Item, VARIANT FAR* Cancel)
{
  V_VT( Cancel ) = VT_BOOL;
  V_BOOL( Cancel ) = VARIANT_TRUE;
}
```

The following VB.NET sample cancels expanding or collapsing items:

```
Private Sub AxGrid1_BeforeExpandItem(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_BeforeExpandItemEvent) Handles AxGrid1.BeforeExpandItem
  e.cancel = True
End Sub
```

The following C# sample cancels expanding or collapsing items:

```
private void axGrid1_BeforeExpandItem(object sender,
AxEXGRIDLib._IGridEvents_BeforeExpandItemEvent e)
{
  e.cancel = true;
}
```

The following VFP sample cancels expanding or collapsing items:

```
*** ActiveX Control Event ***
LPARAMETERS item, cancel

cancel = .t.
```

event ButtonClick (Item as HITEM, ColIndex as Long, Key as Variant)

Occurs when user clicks on the cell's button.

Type	Description
Item as HITEM	A long expression that determines the item's handle. If the Item parameter is 0, and the ColIndex property is different than zero, the ColIndex indicates the handle of the cell where the state is changed.
ColIndex as Long	A long expression that indicates the column's index, if the Item parameter is not zero, a long expression that indicates the handle of the cell if the Item parameter is 0.
Key as Variant	Specifies the button's key that's clicked. If the Key parameter is empty, the user clicked the drop down button of the editor.

Use the ButtonClick event to notify your application that a button is clicked. Use the ColumnClick event to notify your application that the user clicks the column's header. Use the CellImageClick event to notify your application that the user clicks an icon in the cell. You can assign a button to a cell using any of the following ways:

- The CellHasButton property specifies whether the cell displays a button. Use the CellValue property indicates the button's caption. In this case the Key parameter is empty.
- The AddButton method adds a button to an editor. The Key parameter indicates the key of the button being clicked. A drop down type editor like ButtonType, DropDownType, DropDownListType, PickEditType, DateType, ColorType, FontType and PictureType includes a drop down button. The Key parameter is empty, for a drop down button.

Syntax for ButtonClick event, **/NET** version, on:

C#

```
private void ButtonClick(object sender,int Item,int ColIndex,object Key)
{
}
```

VB

```
Private Sub ButtonClick(ByVal sender As System.Object,ByVal Item As Integer,ByVal
ColIndex As Integer,ByVal Key As Object) Handles ButtonClick
End Sub
```

Syntax for ButtonClick event, **/COM** version, on:

C#

```
private void ButtonClick(object sender,
AxEXGRIDLib._IGridEvents_ButtonClickEvent e)
{
}
```

C++

```
void OnButtonClick(long Item,long ColIndex,VARIANT Key)
{
}
```

C++ Builder

```
void __fastcall ButtonClick(TObject *Sender,Exgridlib_tlb::HITEM Item,long
ColIndex,Variant Key)
{
}
```

Delphi

```
procedure ButtonClick(ASender: TObject; Item : HITEM;ColIndex : Integer;Key :
OleVariant);
begin
end;
```

Delphi 8 (.NET only)

```
procedure ButtonClick(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_ButtonClickEvent);
begin
end;
```

Powe…

```
begin event ButtonClick(long Item,long ColIndex,any Key)
end event ButtonClick
```

VB.NET

```
Private Sub ButtonClick(ByVal sender As System.Object, ByVal e As
AxEXGRIDLib._IGridEvents_ButtonClickEvent) Handles ButtonClick
End Sub
```

VB6

```
Private Sub ButtonClick(ByVal Item As EXGRIDLibCtl.HITEM,ByVal ColIndex As
Long,ByVal Key As Variant)
End Sub
```

VBA

```
Private Sub ButtonClick(ByVal Item As Long,ByVal ColIndex As Long,ByVal Key As
Variant)
```

```
End Sub
```

VFP

```
LPARAMETERS Item,ColIndex,Key
```

Xbas…

```
PROCEDURE OnButtonClick(oGrid,Item,ColIndex,Key)
RETURN
```

Syntax for ButtonClick event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="ButtonClick(Item,ColIndex,Key)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function ButtonClick(Item,ColIndex,Key)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComButtonClick HITEM llItem Integer llColIndex Variant llKey
   Forward Send OnComButtonClick llItem llColIndex llKey
End_Procedure
```

Visual Objects

```
METHOD OCX_ButtonClick(Item,ColIndex,Key) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_ButtonClick(int _Item,int _ColIndex,COMVariant _Key)
{
}
```

XBasic

```
function ButtonClick as v (Item as OLE::Exontrol.Grid.1::HITEM,ColIndex as N,Key as
A)
end function
```

dBASE

```
function nativeObject_ButtonClick(Item,ColIndex,Key)
return
```

The following VB sample displays the key of the button being clicked:

```
With Grid1.Columns.Add("Editor").Editor
   .EditType = EditType
   .AddButton "Key1", 1
   .AddButton "Key2", 2, EXGRIDLibCtl.AlignmentEnum.RightAlignment, "This is a bit of
text that should be displayed when the cursor is over the button", "Some information"
   .AddButton "Key3", 3, EXGRIDLibCtl.AlignmentEnum.RightAlignment
End With
```

```
...
```

```
Private Sub Grid1_ButtonClick(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As Long,
ByVal Key As Variant)
   ' Displays the button's key that was clicked
   Dim mes As String
   mes = "You have pressed the button"
   mes = mes + Iif(Len(Key) = 0, "", " '" & Key & "'")
   mes = mes + " of cell '" & Grid1.Items.CellValue(Item) & "'."
   Debug.Print mes
End Sub
```

The following VB sample displays the caption of the cell where a button is clicked:

```
Private Sub Grid1_ButtonClick(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As Long,
ByVal Key As Variant)
   With Grid1.Items
      Debug.Print .CellCaption(Item, ColIndex) & ", Key = '" & Key & "'"
   End With
End Sub
```

The following C++ sample displays the caption of the cell where a button is clicked:

```
#include "Items.h"
void OnButtonClickGrid1(long Item, long ColIndex, const VARIANT FAR& Key)
{
   CItems items = m_grid.GetItems();
   CString strFormat;
   strFormat.Format( "%s, Key = '%s'", items.GetCellCaption( COleVariant( Item ),
COleVariant( ColIndex ) ), V2S( (LPVARIANT)&Key ) );
```

```
    OutputDebugString( strFormat );
}
```

where the V2S string may look like follows:

```
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;
}
```

or

```
static string V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        CComVariant vt;
        if ( SUCCEEDED( vt.ChangeType( VT_BSTR, pv ) ) )
        {
            USES_CONVERSION;
            return OLE2T(V_BSTR( &vt ));
        }
    }
    return szDefault;
}
```

if you are using STL.

The following VB.NET sample displays the caption of the cell where a button is clicked:

```
Private Sub AxGrid1_ButtonClick(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_ButtonClickEvent) Handles AxGrid1.ButtonClick
   With AxGrid1.Items
      Dim strKey As String = ""
      If Not (e.key Is Nothing) Then
         strKey = e.key.ToString()
      End If
      Debug.Print(.CellCaption(e.item, e.colIndex).ToString() + ", Key = " + strKey)
   End With
End Sub
```

The following C# sample displays the caption of the cell where a button is clicked:

```
private void axGrid1_ButtonClick(object sender,
AxEXGRIDLib._IGridEvents_ButtonClickEvent e)
{
   string strKey = "";
   if (e.key != null)
      strKey = e.key.ToString();
   System.Diagnostics.Debug.WriteLine(axGrid1.Items.get_CellCaption(e.item, e.colIndex)
+ ", Key = " + strKey);
}
```

The following VFP sample displays the caption of the cell where a button is clicked:

```
*** ActiveX Control Event ***
LPARAMETERS item, colindex, key

with thisform.Grid1.Items
   .DefaultItem = item
   wait window nowait .CellCaption(0, colindex)
endwith
```

event CellImageClick (Item as HITEM, ColIndex as Long)

Fired after the user clicks on the image's cell area.

Type	Description
Item as HITEM	A long expression that determines the item's handle. If the Item parameter is 0, and the ColIndex property is different than zero, the ColIndex indicates the handle of the cell where the state is changed.
ColIndex as Long	A long expression that indicates the column's index, if the Item parameter is not zero, a long expression that indicates the handle of the cell if the Item parameter is 0.

The CellImageClick event notifies your application that an icon in the cell is clicked. 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 get the index of icon being clicked, if the cell displays multiple icons using the CellImages property. The CellImageClick event is not fired if you are clicking a custom size picture added with the CellPicture property. Use the CellHasCheckBox or CellHasRadioButton property to assign a check box or a radio button to a cell. Use the CellStateChanged event to notify your application that the the cell's checkbox or radio button is clicked:

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,
AxEXGRIDLib._IGridEvents_CellImageClickEvent e)
{
}
```

C++

```
void OnCellImageClick(long Item,long ColIndex)
{
```

```
}
```

C++ Builder

```
void __fastcall CellImageClick(TObject *Sender,Exgridlib_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: AxEXGRIDLib._IGridEvents_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 AxEXGRIDLib._IGridEvents_CellImageClickEvent) Handles CellImageClick
End Sub
```

VB6

```
Private Sub CellImageClick(ByVal Item As EXGRIDLibCtl.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(oGrid,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.Grid.1::HITEM,ColIndex as N)
end function
```

dBASE

```
function nativeObject_CellImageClick(Item,ColIndex)
return
```

The following VB sample changes the cell's icon when the user clicks the icon:

```
Private Sub Grid1_CellImageClick(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As
Long)
   With Grid1.Items
      .CellImage(Item, ColIndex) = (.CellImage(Item, ColIndex) Mod 2) + 1
   End With
End Sub
```

The following VB sample displays the index of icon being clicked, when the cell displays multiple icons (CellImages property):

```
Private Sub Grid1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Dim i As HITEM, h As HitTestInfoEnum, c As Long
   With Grid1
      i = .ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, h)
   End With
   If (i <> 0) Then
      If exHTCellIcon = (h And exHTCellIcon) Then
         Debug.Print "The index of icon being clicked is: " & (h And &HFFFF0000) / 65536
      End If
   End If
End Sub
```

The following C++ sample changes the cell's icon being clicked:

```
#include "Items.h"
void OnCellImageClickGrid1(long Item, long ColIndex)
{
   CItems items = m_grid.GetItems();
   COleVariant vtItem( Item ), vtColumn( ColIndex );
   items.SetCellImage( vtItem , vtColumn , items.GetCellImage( vtItem, vtColumn ) % 2 + 1
);
}
```

The following VB.NET sample changes the cell's icon being clicked:

```
Private Sub AxGrid1_CellImageClick(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_CellImageClickEvent) Handles AxGrid1.CellImageClick
   With AxGrid1.Items
      .CellImage(e.item, e.colIndex) = .CellImage(e.item, e.colIndex) Mod 2 + 1
   End With
End Sub
```

The following C# sample changes the cell's icon being clicked:

```
private void axGrid1_CellImageClick(object sender,
AxEXGRIDLib._IGridEvents_CellImageClickEvent e)
{
   axGrid1.Items.set_CellImage(e.item, e.colIndex, axGrid1.Items.get_CellImage(e.item,
```

```
e.colIndex) % 2 + 1);
}
```

The following VFP sample changes the cell's icon being clicked:

```
*** ActiveX Control Event ***
LPARAMETERS item, colindex

with thisform.Grid1.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 determines the item's handle. If the Item parameter is 0, and the ColIndex property is different than zero, the ColIndex indicates the handle of the cell where the state is changed.
ColIndex as Long	A long expression that indicates the column's index, if the Item parameter is not zero, 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

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.

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,
AxEXGRIDLib._IGridEvents_CellStateChangedEvent e)
{
}
```

C++

```
void OnCellStateChanged(long Item,long ColIndex)
{
}
```

C++ Builder

```
void __fastcall CellStateChanged(TObject *Sender,Exgridlib_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_CellStateChangedEvent) Handles CellStateChanged
End Sub
```

VB6

```
Private Sub CellStateChanged(ByVal Item As EXGRIDLibCtl.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(oGrid,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.Grid.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 Grid1_CellStateChanged(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As Long)
   With Grid1.Items
```

```
    Debug.Print "'" & .CellValue(Item, ColIndex) & "' = " & .CellState(Item, ColIndex)
  End With
End Sub
```

The following C++ sample displays a message when the user clicks a check box or a radio button:

```
#include "Items.h"
void OnCellStateChangedGrid1(long Item, long ColIndex)
{
  CItems items = m_grid.GetItems();
  COleVariant vtItem( Item ), vtColumn( ColIndex );
  CString strFormat;
  strFormat.Format( "'%s' = %i", items.GetCellCaption( vtItem, vtColumn ),
items.GetCellState( vtItem, vtColumn ) );
  OutputDebugString( strFormat );
}
```

The following VB.NET sample displays a message when the user clicks a check box or a radio button:

```
Private Sub AxGrid1_CellStateChanged(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_CellStateChangedEvent) Handles AxGrid1.CellStateChanged
  With AxGrid1.Items
    Debug.Print(.CellCaption(e.item, e.colIndex) & " = " & .CellState(e.item,
e.colIndex).ToString())
  End With
End Sub
```

The following C# sample displays a message when the user clicks a check box or a radio button:

```
private void axGrid1_CellStateChanged(object sender,
AxEXGRIDLib._IGridEvents_CellStateChangedEvent e)
{
  string strOutput = axGrid1.Items.get_CellCaption(e.item, e.colIndex).ToString();
  strOutput += " = " + axGrid1.Items.get_CellState(e.item, e.colIndex).ToString();
  System.Diagnostics.Debug.WriteLine(strOutput);
}
```

The following VFP sample displays a message when the user clicks a check box or a radio button:

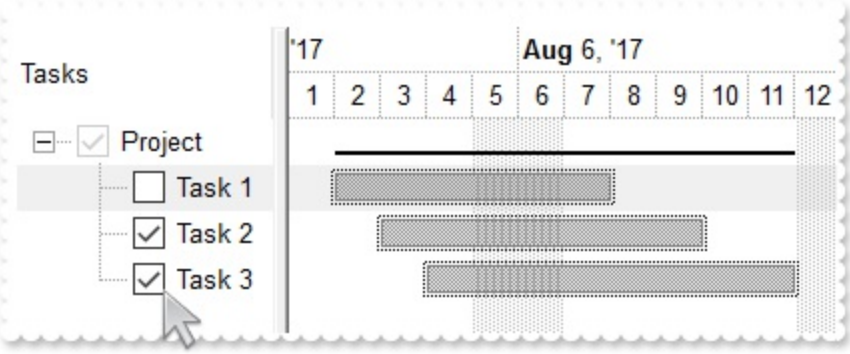
```
*** ActiveX Control Event ***
LPARAMETERS item, colindex

local sOutput
sOutput = ""
with thisform.Grid1.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, 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	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 Grid1_CellStateChanging(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex
As Long, NewState As Long)
  With Grid1
    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,
AxEXGRIDLib._IGridEvents_CellStateChangingEvent e)
{
}
```

C++

```
void OnCellStateChanging(long  Item,long  ColIndex,long FAR*  NewState)
{
}
```

C++ Builder

```
void __fastcall CellStateChanging(TObject *Sender,Exgridlib_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_CellStateChangingEvent) Handles CellStateChanging
End Sub
```

VB6

```
Private Sub CellStateChanging(ByVal Item As EXGRIDLibCtl.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(oGrid,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.Grid.1::HITEM,ColIndex  as
N,NewState  as  N)
end function
```

dBASE

```
function nativeObject_CellStateChanging(Item,ColIndex,NewState)
return
```

event Change (Item as HITEM, ColIndex as Long, NewValue as Variant)

Occurs when the user changes the cell's content.

Type	Description
Item as HITEM	A long expression that determines the item's handle. If the Item parameter is 0, and the ColIndex property is different than zero, the ColIndex indicates the handle of the cell where the state is changed.
ColIndex as Long	A long expression that indicates the column's index, if the Item parameter is not zero, a long expression that indicates the handle of the cell if the Item parameter is 0.
NewValue as Variant	A Variant value that indicates the newly cell's value. You can use the EditingText property to retrieve the caption of the editor while the control is in edit mode.

The Change event notifies your application that the cell's value is about to be changed. The NewValue parameter of the Change event indicates the newly value to be assigned to the cell's value. The EditingText property returns the caption being shown on the editor while the control runs in edit mode. Changing the CellValue property invokes Change event too. During the Change event it is possible to have *recursive calls*, if you are changing the CellValue property (only when you assign a value to a cell, not when you are retrieving the cell's value). If you are changing the *other cell's value*, during the Change event you have to add a C++ code like follows in order to avoid *recursive calls*:

```
static sg_ChangeCounter = 0;
void OnChangeGrid1(long Item, long ColIndex, VARIANT FAR* NewValue)
{
   if ( sg_ChangeCounter == 0)
   {
      sg_ChangeCounter++;
      m_Items.SetCellValue( COleVariant( Item ), COleVariant( (long)othercolumn ),
*NewValue );
      sg_ChangeCounter--;
   }
}
```

or in VB you could have like this:

```
Private sg_ChangeCounter As Long
```

```
Private Sub Grid1_Change(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As Long,
NewValue As Variant)
   If (sg_ChangeCounter = 0) Then
      sg_ChangeCounter = sg_ChangeCounter + 1
      Grid1.Items.CellValue(Item, othercolumn) = NewValue
      sg_ChangeCounter = sg_ChangeCounter - 1
   End If
End Sub
```

The edit events are fired in the following order:

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

Use the CellEditor or Editor property to assign an editor to a cell or to a column. Use the Edit event to notify your application that the editing operation begins. The Change event notifies that the editing focused cell ended. If the control is bounded to an ADO recordset the Change event is automatically called when the user changes the focused cell, and it updates the recordset too. The control fires the ValidateValue event before calling the Change event, if the CauseValidateValue property is True. Please note that the Change event is called also when loading, or adding new items , so you need to use an internal counter (like explained bellow) to avoid calling the Change event during adding or loading the items, if it is not case (increases the iChanging variable before loading items, and decreases the iChanging member when adding items is done). Call the Refresh method, when changing the value for a cell that has the CellSingleLine property on False.

Syntax for Change event, **/NET** version, on:

C#

```
private void Change(object sender,int Item,int ColIndex,ref object NewValue)
{
}
```

VB

```
Private Sub Change(ByVal sender As System.Object,ByVal Item As Integer,ByVal
ColIndex As Integer,ByRef NewValue As Object) Handles Change
End Sub
```

Syntax for Change event, **/COM** version, on:

C#
```
private void Change(object sender, AxEXGRIDLib._IGridEvents_ChangeEvent e)
{
}
```

C++
```
void OnChange(long Item,long ColIndex,VARIANT FAR* NewValue)
{
}
```

C++ Builder
```
void __fastcall Change(TObject *Sender,Exgridlib_tlb::HITEM Item,long
ColIndex,Variant * NewValue)
{
}
```

Delphi
```
procedure Change(ASender: TObject; Item : HITEM;ColIndex : Integer;var
NewValue : OleVariant);
begin
end;
```

Delphi 8 (.NET only)
```
procedure Change(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_ChangeEvent);
begin
end;
```

Powe…
```
begin event Change(long Item,long ColIndex,any NewValue)
end event Change
```

VB.NET
```
Private Sub Change(ByVal sender As System.Object, ByVal e As
AxEXGRIDLib._IGridEvents_ChangeEvent) Handles Change
End Sub
```

VB6
```
Private Sub Change(ByVal Item As EXGRIDLibCtl.HITEM,ByVal ColIndex As
Long,NewValue As Variant)
End Sub
```

VBA
```
Private Sub Change(ByVal Item As Long,ByVal ColIndex As Long,NewValue As
Variant)
```

```
End Sub
```

VFP

```
LPARAMETERS Item,ColIndex,NewValue
```

Xbas…

```
PROCEDURE OnChange(oGrid,Item,ColIndex,NewValue)
RETURN
```

Syntax for Change event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="Change(Item,ColIndex,NewValue)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function Change(Item,ColIndex,NewValue)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComChange HITEM llItem Integer llColIndex Variant llNewValue
   Forward Send OnComChange llItem llColIndex llNewValue
End_Procedure
```

Visual Objects

```
METHOD OCX_Change(Item,ColIndex,NewValue) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_Change(int _Item,int _ColIndex,COMVariant /*variant*/ _NewValue)
{
}
```

XBasic

```
function Change as v (Item as OLE::Exontrol.Grid.1::HITEM,ColIndex as N,NewValue
as A)
end function
```

dBASE

```
function nativeObject_Change(Item,ColIndex,NewValue)
return
```

The following VB sample displays the newly value of the focused cell:

```
Private Sub Grid1_Change(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As Long,
NewValue As Variant)
   ' Displays the old/new cell's value
   Debug.Print "The current cell's value is '" & Grid1.Items.CellValue(Item, ColIndex) & "'."
   Debug.Print "The newly cell's value is '" & NewValue & "'."
End Sub
```

You can change the newly cell's value by changing the NewValue parameter of the Change event. If you are changing the CellValue property during the Change event a recursive calls occurs, so you need to protect recursive calls using an internal counter that's increased when Change event starts, and decreased when the Change event ends like in the following VB sample:

```
Private iChanging As Long

Private Sub Grid1_Change(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As Long,
NewValue As Variant)
   If (iChanging = 0) Then
      iChanging = iChanging + 1
        ' here's safe to change the Items.CellValue property
      iChanging = iChanging - 1
   End If
End Sub
```

The following sample is the C++ equivalent:

```
long iChanging = 0;
void OnChangeGrid1(long Item, long ColIndex, VARIANT FAR* NewValue)
{
   if ( iChanging == 0 )
   {
      iChanging++;
      // here's safe to call Items.CellValue property, to avoid recursive calls.
      iChanging--;
   }
}
```

The following C++ sample displays the newly value of the focused cell:

```
#include "Items.h"
void OnChangeGrid1(long Item, long ColIndex, VARIANT FAR* NewValue)
{
   if ( ::IsWindow( m_grid.m_hWnd ) )
   {
      CItems items = m_grid.GetItems();
      COleVariant vtItem( Item ), vtColumn( ColIndex );
      CString strFormat;
      strFormat.Format( "'%s' = %s", V2S( &items.GetCellValue( vtItem, vtColumn ) ), V2S(
NewValue ) );
      OutputDebugString( strFormat );
   }
}
```

where the V2S function converts a VARIANT to a string value, and may look like follows:

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
   if ( pv )
   {
      if ( pv->vt == VT_ERROR )
         return szDefault;

      COleVariant vt;
      vt.ChangeType( VT_BSTR, pv );
      return V_BSTR( &vt );
   }
   return szDefault;
}
```

The following VB.NET sample displays the newly value of the focused cell:

```
Private Sub AxGrid1_Change(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_ChangeEvent) Handles AxGrid1.Change
   With AxGrid1.Items
      Debug.Print("Old Value: " & .CellValue(e.item, e.colIndex) & " New Value " &
e.newValue.ToString())
   End With
```

```
End Sub
```

The following C# sample displays the newly value of the focused cell:

```
private void axGrid1_Change(object sender, AxEXGRIDLib._IGridEvents_ChangeEvent e)
{
   System.Diagnostics.Debug.WriteLine("Old Value " + axGrid1.Items.get_CellValue(e.item,
e.colIndex).ToString() + " New Value " + e.newValue.ToString());
}
```

The following VFP sample displays the newly value of the focused cell:

```
*** ActiveX Control Event ***
LPARAMETERS item, colindex, newvalue

with thisform.Grid1.Items
   .DefaultItem = item
   local oldvalue
   oldvalue = .CellValue(0,colindex)
   wait window nowait "Old Value " + str(oldvalue)
   wait window nowait "New Value " + str(newvalue)
endwith
```

event Click ()

Occurs when the user presses and then releases the left mouse button over the Grid 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. If you have a column of hyperlink cells (CellHyperLink property is True) you should use HyperLinkClick event. The AnchorClick event notifies your application that the user clicks an <a> anchor element.

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(oGrid)
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 whose header has been clicked.

The ColumnClick event is fired when the user clicks the column's header. By default, the control sorts by the column when user clicks the column's header. Use the SortOnClick property to specify the operation that control does when user clicks the column's caption. Use the ColumnFromPoint property to access the column from point. Use the ItemFromPoint property to access the item from point. 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.EXGRIDLib.Column Column)
{
}
```

VB
```
Private Sub ColumnClick(ByVal sender As System.Object,ByVal Column As
exontrol.EXGRIDLib.Column) Handles ColumnClick
End Sub
```

Syntax for ColumnClick event, **/COM** version, on:

C#
```
private void ColumnClick(object sender,
AxEXGRIDLib._IGridEvents_ColumnClickEvent e)
{
}
```

C++
```
void OnColumnClick(LPDISPATCH Column)
{
}
```

C++ Builder
```
void __fastcall ColumnClick(TObject *Sender,Exgridlib_tlb::IColumn *Column)
{
}
```

Delphi
```
procedure ColumnClick(ASender: TObject; Column : IColumn);
```

```
begin
end;
```

Delphi 8 (.NET only)

```
procedure ColumnClick(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_ColumnClickEvent) Handles ColumnClick
End Sub
```

VB6

```
Private Sub ColumnClick(ByVal Column As EXGRIDLibCtl.IColumn)
End Sub
```

VBA

```
Private Sub ColumnClick(ByVal Column As Object)
End Sub
```

VFP

```
LPARAMETERS Column
```

Xbas…

```
PROCEDURE OnColumnClick(oGrid,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.Grid.1::IColumn)
end function
```

dBASE

```
function nativeObject_ColumnClick(Column)
return
```

The following VB sample displays the caption of the column being clicked:

```
Private Sub Grid1_ColumnClick(ByVal Column As EXGRIDLibCtl.IColumn)
   Debug.Print Column.Caption
End Sub
```

The following C++ sample displays the caption of the column being clicked:

```
#include "Column.h"
void OnColumnClickGrid1(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 AxGrid1_ColumnClick(ByVal sender As Object, ByVal e As
```

```
AxEXGRIDLib._IGridEvents_ColumnClickEvent) Handles AxGrid1.ColumnClick
    MessageBox.Show(e.column.Caption)
End Sub
```

The following C# sample displays the caption of the column being clicked:

```
private void axGrid1_ColumnClick(object sender,
AxEXGRIDLib._IGridEvents_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 double-clicks 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, AxEXGRIDLib._IGridEvents_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_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(oGrid,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.Grid.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.Grid.1::OLE_YPOS_PIXELS)
end function
```

dBASE

```
function nativeObject_DblClick(Shift,X,Y)
return
```

The following VB sample displays a message when the user double clicks an item:

```
Private Sub Grid1_DblClick(Shift As Integer, X As Single, Y As Single)
  With Grid1
    ' 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 = .ItemFromPoint(X, Y, c, hit)
    If Not (h = 0) Or Not (c = 0) Then
      MsgBox "The '" & .Items.CellValue(h, c) & "' item has been double clicked."
```

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

The following VB sample use DblClick event to start editing control, if the AutoEdit property is False.

```
Private Sub Grid1_DblClick(Shift As Integer, X As Single, Y As Single)
  ' Starts the editing operation if the user dbl click in the control
  With Grid1
    If (Not .AutoEdit) Then
      ' Did user dbl click on an item?
      Dim c As Long, hit as Long
      If Not (.ItemFromPoint(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, hit) =
0) Then
        .Edit
      End If
    End If
  End With
End Sub
```

The following C++ sample displays the value of the cell being double clicked (including the inner cells):

```
#include "Items.h"
void OnDblClickGrid1(short Shift, long X, long Y)
{
  long c = NULL, hit = NULL;
  long h = m_grid.GetItemFromPoint( X, Y, &c, &hit );
  if ( ( h != 0 ) || ( c != 0 ) )
  {
    COleVariant vtItem( h ), vtColumn( c );
    CString strCaption = V2S( &m_grid.GetItems().GetCellValue( vtItem, vtColumn ) );
    MessageBox( strCaption );
  }
}
```

where the V2S function converts a VARIANT value to a string, and may look like follows:

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
   if ( pv )
   {
      if ( pv->vt == VT_ERROR )
         return szDefault;

      COleVariant vt;
      vt.ChangeType( VT_BSTR, pv );
      return V_BSTR( &vt );
   }
   return szDefault;
}
```

The following VB.NET sample displays the value of the cell being double clicked (including the inner cells):

```
Private Sub AxGrid1_DblClick(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_DblClickEvent) Handles AxGrid1.DblClick
   Dim h As Integer, c As Integer, hit As EXGRIDLib.HitTestInfoEnum
   With AxGrid1
      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 value of the cell being double clicked (including the inner cells):

```
private void axGrid1_DblClick(object sender, AxEXGRIDLib._IGridEvents_DblClickEvent e)
{
   EXGRIDLib.HitTestInfoEnum hit;
   int c = 0, h = axGrid1.get_ItemFromPoint(e.x, e.y, out c, out hit);
   if ((h != 0) || (c != 0))
      MessageBox.Show(axGrid1.Items.get_CellCaption(h, c).ToString());
}
```

The following VFP sample displays the value of the cell being double clicked:

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

local c, hit
c = 0
hit = 0

with thisform.Grid1
   .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
   if ( .Items.DefaultItem != 0 )
      wait window nowait .Items.CellCaption( 0, c )
   endif
endwith
```

event Edit (Item as HITEM, ColIndex as Long, Cancel as Boolean)

Occurs just before editing the focused cell.

Type	Description
Item as HITEM	A long expression that determines the item's handle. If the Item parameter is 0, and the ColIndex property is different than zero, the ColIndex indicates the handle of the cell where the state is changed.
ColIndex as Long	A long expression that indicates the column's index, if the Item parameter is not zero, a long expression that indicates the handle of the cell if the Item parameter is 0.
Cancel as Boolean	A boolean expression that indicates whether the editing operation is canceled.

The Edit event is fired when the edit operation is about to begin. Use the Edit event to disable editing specific cells. The Edit event is not fired if the user changes programmatically the CellValue property. Use the EditOpen event to notify your application that editing the cell started. Use the EditClose event to notify your application that editing the cell ended. Use the Change event to notify your application that user changes the cell's value. Use the Edit method to edit a cell by code. Use the CellEditor or Editor property to assign an editor to a cell or to a column.

The edit events are fired in the following order:

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

Syntax for Edit event, **/NET** version, on:

C#
```
private void EditEvent(object sender,int Item,int ColIndex,ref bool Cancel)
{
}
```

VB
```
Private Sub EditEvent(ByVal sender As System.Object,ByVal Item As Integer,ByVal ColIndex As Integer,ByRef Cancel As Boolean) Handles EditEvent
```

```
End Sub
```

Syntax for Edit event, **/COM** version, on:

C#

```
private void EditEvent(object sender, AxEXGRIDLib._IGridEvents_EditEvent e)
{
}
```

C++

```
void OnEdit(long Item,long ColIndex,BOOL FAR* Cancel)
{
}
```

C++ Builder

```
void __fastcall Edit(TObject *Sender,Exgridlib_tlb::HITEM Item,long
ColIndex,VARIANT_BOOL * Cancel)
{
}
```

Delphi

```
procedure Edit(ASender: TObject; Item : HITEM;ColIndex : Integer;var Cancel :
WordBool);
begin
end;
```

Delphi 8 (.NET only)

```
procedure EditEvent(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_EditEvent);
begin
end;
```

Powe…

```
begin event Edit(long Item,long ColIndex,boolean Cancel)
end event Edit
```

VB.NET

```
Private Sub EditEvent(ByVal sender As System.Object, ByVal e As
AxEXGRIDLib._IGridEvents_EditEvent) Handles EditEvent
End Sub
```

VB6

```
Private Sub Edit(ByVal Item As EXGRIDLibCtl.HITEM,ByVal ColIndex As Long,Cancel
As Boolean)
End Sub
```

VBA

```
Private Sub Edit(ByVal Item As Long,ByVal ColIndex As Long,Cancel As Boolean)
End Sub
```

VFP

```
LPARAMETERS Item,ColIndex,Cancel
```

Xbas…

```
PROCEDURE OnEdit(oGrid,Item,ColIndex,Cancel)
RETURN
```

Syntax for Edit event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="Edit(Item,ColIndex,Cancel)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function Edit(Item,ColIndex,Cancel)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComEdit HITEM llItem Integer llColIndex Boolean llCancel
   Forward Send OnComEdit llItem llColIndex llCancel
End_Procedure
```

Visual Objects

```
METHOD OCX_Edit(Item,ColIndex,Cancel) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_Edit(int _Item,int _ColIndex,COMVariant /*bool*/ _Cancel)
{
}
```

XBasic

```
function Edit as v (Item as OLE::Exontrol.Grid.1::HITEM,ColIndex as N,Cancel as L)
end function
```

dBASE

```
function nativeObject_Edit(Item,ColIndex,Cancel)
return
```

The following VB sample disables editing cells in the first column:

```
Private Sub Grid1_Edit(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As Long, Cancel As Boolean)
   ' Cancels editing first column
   Cancel = IIf(ColIndex = 0, True, False)
End Sub
```

The following VB sample changes the cell's value to a default value, if the user enters an empty value:

```
Private Sub Grid1_Edit(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As Long, Cancel As Boolean)
   ' Sets the 'default' value for empty cell
   With Grid1.Items
      If (Len(.CellValue(Item, ColIndex)) = 0) Then
         .CellValue(Item, ColIndex) = "default"
      End If
   End With
End Sub
```

The following C++ sample disables editing cells in the first column:

```
void OnEditGrid1(long Item, long ColIndex, BOOL FAR* Cancel)
{
   if ( ColIndex == 0 )
      *Cancel = TRUE;
}
```

The following VB.NET sample disables editing cells in the first column:

```
Private Sub AxGrid1_EditEvent(ByVal sender As Object, ByVal e As AxEXGRIDLib._IGridEvents_EditEvent) Handles AxGrid1.EditEvent
   If (e.colIndex = 0) Then
      e.cancel = True
   End If
End Sub
```

The following C# sample disables editing cells in the first column:

```
private void axGrid1_EditEvent(object sender, AxEXGRIDLib._IGridEvents_EditEvent e)
{
    if (e.colIndex == 0)
        e.cancel = true;
}
```

The following VFP sample disables editing cells in the first column:

```
*** ActiveX Control Event ***
LPARAMETERS item, colindex, cancel

if ( colindex = 0 )
    cancel = .t.
endif
```

event EditClose ()

Occurs when the edit operation ends.

Type	Description

Use the EditClose event to notify your application that the editor is closed. The EditClose event is fired when the focused cell ends editing. Use the FocusItem property to determine the handle of the item where the edit operation ends. Use the FocusColumnIndex property to determine the index of the column where the edit operation ends. The Editing specifies the window's handle of the built-in editor while the control is running in edit mode. The EditingText property returns the caption being shown on the editor while the control runs in edit mode. Use the EditClose method to closes the current editor, by code. For instance, the EditClose event is not fired when user hides the drop down portion of the editor. Use the Edit event to prevent editing a cell.

The edit events are fired in the following order:

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

Syntax for EditClose event, **/NET** version, on:

C#

```
private void EditCloseEvent(object sender)
{
}
```

VB

```
Private Sub EditCloseEvent(ByVal sender As System.Object) Handles
EditCloseEvent
End Sub
```

Syntax for EditClose event, **/COM** version, on:

C#

```
private void EditCloseEvent(object sender, EventArgs e)
{
}
```

C++

```
void OnEditClose()
{
}
```

C++ Builder

```
void __fastcall EditClose(TObject *Sender)
{
}
```

Delphi

```
procedure EditClose(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)

```
procedure EditCloseEvent(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event EditClose()
end event EditClose
```

VB.NET

```
Private Sub EditCloseEvent(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles EditCloseEvent
End Sub
```

VB6

```
Private Sub EditClose()
End Sub
```

VBA

```
Private Sub EditClose()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnEditClose(oGrid)
RETURN
```

Syntax for EditClose event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="EditClose()" LANGUAGE="JScript">
```

```
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function EditClose()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComEditClose
   Forward Send OnComEditClose
End_Procedure
```

Visual Objects

```
METHOD OCX_EditClose() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_EditClose()
{
}
```

XBasic

```
function EditClose as v ()
end function
```

dBASE

```
function nativeObject_EditClose()
return
```

The following VB sample displays the window's handle of the built-in editor being closed:

```
Private Sub Grid1_EditClose()
   Debug.Print "EditClose " & Grid1.Editing
End Sub
```

The following VB sample displays the caption of the cell where the edit operation ends:

```
Private Sub Grid1_EditClose()
   With Grid1.Items
      Debug.Print "EditClose on '"; .CellCaption(.FocusItem, Grid1.FocusColumnIndex) & "'."
   End With
End Sub
```

The following C++ sample displays the handle of the built-in editor being closed:

```
#include "Items.h"
void OnEditCloseGrid1()
{
   CItems items = m_grid.GetItems();
   COleVariant vtItem( items.GetFocusItem() ), vtColumn( m_grid.GetFocusColumnIndex() );
   CString strFormat;
   strFormat.Format( "'%s'  %i", V2S( &items.GetCellValue( vtItem, vtColumn ) ),
m_grid.GetEditing() );
   OutputDebugString( strFormat );
}
```

The following VB.NET sample displays the handle of the built-in editor being closed:

```
Private Sub AxGrid1_EditCloseEvent(ByVal sender As Object, ByVal e As System.EventArgs)
Handles AxGrid1.EditCloseEvent
   With AxGrid1
      Debug.Print(.Items.CellValue(.Items.FocusItem, .FocusColumnIndex) & " " &
.Editing.ToString())
   End With
End Sub
```

The following C# sample displays the handle of the built-in editor being closed:

```
private void axGrid1_EditCloseEvent(object sender, EventArgs e)
{
   object cellValue = axGrid1.Items.get_CellValue(axGrid1.Items.FocusItem,
axGrid1.FocusColumnIndex);
   string strOutput = "'" + (cellValue != null ? cellValue.ToString() : "") + "' " +
axGrid1.Editing.ToString();
   System.Diagnostics.Debug.WriteLine( strOutput );
}
```

The following VFP sample displays the handle of the built-in editor being closed:

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

with thisform.Grid1.Items
```

```
    .DefaultItem = .FocusItem()
    wait window nowait str(.CellValue( 0, thisform.Grid1.FocusColumnIndex() ))
    wait window nowait str(thisform.Grid1.Editing())
endwith
```

event EditOpen ()

Occurs when edit operation starts.

Type	Description

Use the EditOpen event to notify your application that the cell's editor is shown and ready to edit the cell. The Editing specifies the window's handle of the built-in editor while the control is running in edit mode. The EditingText property returns the caption being shown on the editor while the control runs in edit mode.

The edit events are fired in the following order:

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

Syntax for EditOpen event, **/NET** version, on:

C#
```
private void EditOpen(object sender)
{
}
```

VB
```
Private Sub EditOpen(ByVal sender As System.Object) Handles EditOpen
End Sub
```

Syntax for EditOpen event, **/COM** version, on:

C#
```
private void EditOpen(object sender, EventArgs e)
{
}
```

C++
```
void OnEditOpen()
{
}
```

C++ Builder

```
void __fastcall EditOpen(TObject *Sender)
{
}
```

Delphi

```
procedure EditOpen(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)

```
procedure EditOpen(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event EditOpen()
end event EditOpen
```

VB.NET

```
Private Sub EditOpen(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles EditOpen
End Sub
```

VB6

```
Private Sub EditOpen()
End Sub
```

VBA

```
Private Sub EditOpen()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnEditOpen(oGrid)
RETURN
```

Syntax for EditOpen event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="EditOpen()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function EditOpen()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComEditOpen
    Forward Send OnComEditOpen
End_Procedure
```

Visual Objects

```
METHOD OCX_EditOpen() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_EditOpen()
{
}
```

XBasic

```
function EditOpen as v ()
end function
```

dBASE

```
function nativeObject_EditOpen()
return
```

The following VB sample unselects the text when the editor is opened (in case the editor is an edit control) (by default, the control selects the text when editor is opened). The following VB sample unselects the text when the editor is opened, by sending an EM_SETSEL message to the cell's edit control:

```
Private Declare Function PostMessage Lib "user32" Alias "PostMessageA" (ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
Private Const EM_SETSEL = &HB1

Private Sub Grid1_EditOpen()
   PostMessage Grid1.Editing, EM_SETSEL, 0, 0
End Sub
```

The following C++ sample unselects the text, when the editor is opened (in case the editor is an edit control):

```
void OnEditOpenGrid1()
{
   ::PostMessage( (HWND)m_grid.GetEditing(), EM_SETSEL, 0, 0 );
}
```

The following VB sample determines the handle of the edit control, without using the Editing property.

```
Private Function getEditWnd(ByVal g As EXGRIDLibCtl.Grid) As Long
   Dim h As Long
   h = GetWindow(g.hwnd, GW_CHILD)
   While Not (h = 0)
      If (getWndClass(h) = "HolderBuiltIn") Then
            getEditWnd = GetWindow(h, GW_CHILD)
         Exit Function
      End If
      h = GetWindow(h, GW_HWNDNEXT)
   Wend
   getEditWnd = 0
End Function

Private Function getWndClass(ByVal h As Long) As String
   Dim s As String
   s = Space(1024)
   GetClassName h, s, 1024
   getWndClass = To0(s)
End Function

Private Function To0(ByVal s As String) As String
   To0 = Left$(s, InStr(s, Chr$(0)) - 1)
End Function

Private Sub Grid1_EditOpen()
   PostMessage getEditWnd(Grid1), EM_SETSEL, 0, 0
End Sub
```

The VB sample requires the following declarations:

```
Private Declare Function PostMessage Lib "user32" Alias "PostMessageA" (ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
Private Declare Function GetWindow Lib "user32" (ByVal hwnd As Long, ByVal wCmd As Long) As Long
Private Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" (ByVal hwnd As Long, ByVal lpString As String, ByVal cch As Long) As Long
Private Declare Function GetClassName Lib "user32" Alias "GetClassNameA" (ByVal hwnd As Long, ByVal lpClassName As String, ByVal nMaxCount As Long) As Long
Private Const GW_CHILD = 5
Private Const GW_HWNDNEXT = 2
Private Const EM_SETSEL = &HB1
```

event Error (Error as Long, Description as String)

Fired when an internal error occurs.

Type	Description
Error as Long	A long expression that indicates the error number.
Description as String	A string expression that describes the error.

The Error event is fired each time when an internal error occurs. The Error event is usually fired when the control is bounded to an ADO Recordset. For instance, if the user changes a field, the control tries to update the current record. If it fails, the Error event is fired. Use the DataSource property to bind the control to a database.

Syntax for Error event, **/NET** version, on:

C#
```
private void Error(object sender,int Err,string Description)
{
}
```

VB
```
Private Sub Error(ByVal sender As System.Object,ByVal Err As Integer,ByVal
Description As String) Handles Error
End Sub
```

Syntax for Error event, **/COM** version, on:

C#
```
private void Error(object sender, AxEXGRIDLib._IGridEvents_ErrorEvent e)
{
}
```

C++
```
void OnError(long Error,LPCTSTR Description)
{
}
```

C++ Builder
```
void __fastcall Error(TObject *Sender,long Error,BSTR Description)
{
}
```

Delphi
```
procedure Error(ASender: TObject; Error : Integer;Description : WideString);
begin
end;
```

Delphi 8 (.NET only)

```
procedure Error(sender: System.Object; e: AxEXGRIDLib._IGridEvents_ErrorEvent);
begin
end;
```

Powe…

```
begin event Error(long Error,string Description)
end event Error
```

VB.NET

```
Private Sub Error(ByVal sender As System.Object, ByVal e As
AxEXGRIDLib._IGridEvents_ErrorEvent) Handles Error
End Sub
```

VB6

```
Private Sub Error(ByVal Error As Long,ByVal Description As String)
End Sub
```

VBA

```
Private Sub Error(ByVal Error As Long,ByVal Description As String)
End Sub
```

VFP

```
LPARAMETERS Error,Description
```

Xbas…

```
PROCEDURE OnError(oGrid,Error,Description)
RETURN
```

Syntax for Error event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="Error(Error,Description)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function Error(Error,Description)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComError Integer llError String llDescription
    Forward Send OnComError llError llDescription
End_Procedure
```

Visual Objects

```
METHOD OCX_Error(Error,Description) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_Error(int _Error,str _Description)
{
}
```

XBasic

```
function Error as v (Error as N,Description as C)
end function
```

dBASE

```
function nativeObject_Error(Error,Description)
return
```

event Event (EventID as Long)

Notifies the application once the control fires an event.

Type	Description
EventID as Long	A Long expression that specifies the identifier of the event. Each internal event of the control has an unique identifier. Use the EventParam(-2) to display entire information about fired event (such as name, identifier, and properties). The EventParam(-1) retrieves the number of parameters of fired event

The Event notification occurs ANY time the control fires an event. *This is useful for X++, which does not support event with parameters passed by reference. Also, this could be useful for C++ Builder or Delphi, which does not handle properly the events with parameters of VARIANT type.*

In X++ the "Error executing code: FormActiveXControl (data source), method ... called with invalid parameters" occurs when handling events that have parameters passed by reference. Passed by reference, means that in the event handler, you can change the value for that parameter, and so the control will takes the new value, and use it. The X++ is NOT able to handle properly events with parameters by reference, so we have the solution.

The solution is using and handling the Event notification and EventParam method., instead handling the event that gives the "invalid parameters" error executing code.

If you are not familiar with what a type library means just handle the Event of the control as follows:

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
   print exgrid1.EventParam(-2).toString();
}
```

This code allows you to display the information for each event of the control being fired as in the list bellow:

```
"MouseMove/-606( 1 , 0 , 145 , 36 )" VT_BSTR
"BeforeDrawPart/54( 2 , -1962866148 , =0 , =0 , =0 , =0 , =false )" VT_BSTR
"AfterDrawPart/55( 2 , -1962866148 , 0 , 0 , 0 , 0 )" VT_BSTR
"MouseMove/-606( 1 , 0 , 145 , 35 )" VT_BSTR
```

Each line indicates an event, and the following information is provided: the name of the event, its identifier, and the list of parameters being passed to the event. The parameters that starts with = character, indicates a parameter by reference, in other words one that can changed during the event handler.

In conclusion, anytime the X++ fires the "invalid parameters." while handling an event, you can use and handle the Event notification and EventParam methods of the control

Syntax for Event event, **/NET** version, on:

C#
```
private void Event(object sender,int EventID)
{
}
```

VB
```
Private Sub Event(ByVal sender As System.Object,ByVal EventID As Integer)
Handles Event
End Sub
```

Syntax for Event event, **/COM** version, on:

C#
```
private void Event(object sender, AxEXGRIDLib._IGridEvents_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: AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_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(oGrid,EventID)
RETURN
```

Syntax for Event event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="Event(EventID)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function Event(EventID)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComEvent Integer llEventID
   Forward Send OnComEvent llEventID
End_Procedure
```

Visual Objects

```
METHOD OCX_Event(EventID) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_Event(int _EventID)
{
}
```

XBasic

```
function Event as v (EventID as N)
end function
```

dBASE

```
function nativeObject_Event(EventID)
return
```

event FilterChange ()

Occurs when filter was changed.

Type	Description

Use the FilterChange event to notify your application that the control's filter is changed. The FilterChanging event occurs just before applying the filter. Use the Filter and FilterType properties to retrieve the column's filter string, if case, and the column's filter type. The ApplyFilter and ClearFilter methods fire the FilterChange event. Use the DisplayFilterButton property to add a filter bar button to the column's caption. Use the FilterBarHeight property to specify the height of the control's filter bar. Use the FilterBarFont property to specify the font for the control's filter bar.

Syntax for FilterChange event, **/NET** version, on:

C#

```
private void FilterChange(object sender)
{
}
```

VB

```
Private Sub FilterChange(ByVal sender As System.Object) Handles FilterChange
End Sub
```

Syntax for FilterChange event, **/COM** version, on:

C#

```
private void FilterChange(object sender, EventArgs e)
{
}
```

C++

```
void OnFilterChange()
{
}
```

C++ Builder

```
void __fastcall FilterChange(TObject *Sender)
{
}
```

Delphi

```
procedure FilterChange(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)

```
procedure FilterChange(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event FilterChange()
end event FilterChange
```

VB.NET

```
Private Sub FilterChange(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles FilterChange
End Sub
```

VB6

```
Private Sub FilterChange()
End Sub
```

VBA

```
Private Sub FilterChange()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnFilterChange(oGrid)
RETURN
```

Syntax for FilterChange event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="FilterChange()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function FilterChange()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComFilterChange
   Forward Send OnComFilterChange
```

```
End_Procedure
```

Visual Objects

```
METHOD OCX_FilterChange() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_FilterChange()
{
}
```

XBasic

```
function FilterChange as v ()
end function
```

dBASE

```
function nativeObject_FilterChange()
return
```

event FilterChanging ()

Notifies your application that the filter is about to change.

Type	Description

The FilterChanging event occurs just before applying the filter. The FilterChange event occurs once the filter is applied, so the list gets filtered. Use the Filter and FilterType properties to retrieve the column's filter string, if case, and the column's filter type. The ApplyFilter and ClearFilter methods fire the FilterChange event. Use the DisplayFilterButton property to add a filter bar button to the column's caption. Use the FilterBarHeight property to specify the height of the control's filter bar. Use the FilterBarFont property to specify the font for the control's filter bar. For instance, you can use the FilterChanging event to start a timer, and count the time to get the filter applied, when the FilterChange event is fired.

Syntax for FilterChanging event, **/NET** version, on:

C#

```
private void FilterChanging(object sender)
{
}
```

VB

```
Private Sub FilterChanging(ByVal sender As System.Object) Handles
FilterChanging
End Sub
```

Syntax for FilterChanging event, **/COM** version, on:

C#

```
private void FilterChanging(object sender, EventArgs e)
{
}
```

C++

```
void OnFilterChanging()
{
}
```

C++ Builder

```
void __fastcall FilterChanging(TObject *Sender)
{
}
```

Delphi

```
procedure FilterChanging(ASender: TObject; );
begin
```

```
end;
```

Delphi 8 (.NET only)

```
procedure FilterChanging(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event FilterChanging()
end event FilterChanging
```

VB.NET

```
Private Sub FilterChanging(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles FilterChanging
End Sub
```

VB6

```
Private Sub FilterChanging()
End Sub
```

VBA

```
Private Sub FilterChanging()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnFilterChanging(oGrid)
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 FocusChanged ()

Occurs when a new cell is focused.

Type	Description

Use the FocusChanged event to notify whether the cell gets the focus. Use the SelectionChanged event to notify your application user changes the selection. Use the FocusColumnIndex property to get to index of the column that has the focus. The FocusItem property indicates the focused item. The SelectColumnInner property indicates the index of an inner cell that has the focus. Use the FullRowSelect property to specify whether the control selects the entire item or a single/multiple cells. Use the SelectItem property to programmatically select an item(it changes the focused items too). Use the InnerCell property to access an inner cell.

The control fires the FocusChanged event when the user changes:

- the focused item
- the focused column or an inner cell gets the focus.

Syntax for FocusChanged event, **/NET** version, on:

C#

```
private void FocusChanged(object sender)
{
}
```

VB

```
Private Sub FocusChanged(ByVal sender As System.Object) Handles
FocusChanged
End Sub
```

Syntax for FocusChanged event, **/COM** version, on:

C#

```
private void FocusChanged(object sender, EventArgs e)
{
}
```

C++

```
void OnFocusChanged()
{
}
```

C++ Builder

```
void __fastcall FocusChanged(TObject *Sender)
{
```

```
}
```

Delphi

```
procedure FocusChanged(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)

```
procedure FocusChanged(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event FocusChanged()
end event FocusChanged
```

VB.NET

```
Private Sub FocusChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles FocusChanged
End Sub
```

VB6

```
Private Sub FocusChanged()
End Sub
```

VBA

```
Private Sub FocusChanged()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnFocusChanged(oGrid)
RETURN
```

Syntax for FocusChanged event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="FocusChanged()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function FocusChanged()
End Function
</SCRIPT>
```

Visual Data...

```
Procedure OnComFocusChanged
    Forward Send OnComFocusChanged
End_Procedure
```

Visual Objects

```
METHOD OCX_FocusChanged() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_FocusChanged()
{
}
```

XBasic

```
function FocusChanged as v ()
end function
```

dBASE

```
function nativeObject_FocusChanged()
return
```

The following VB sample determines the focused cell:

```
Private Sub Grid1_FocusChanged()
    With Grid1
        Dim hFocusCell As EXGRIDLibCtl.HCELL
        hFocusCell = .Items.ItemCell(.Items.FocusItem, .FocusColumnIndex)
        Debug.Print "Focus = " & .Items.CellCaption(, hFocusCell) & " (" & hFocusCell & ")"
    End With
End Sub
```

The following VB sample determines the focused cell, if the control contains inner cells:

```
Private Sub Grid1_FocusChanged()
    With Grid1
        Dim hFocusCell As EXGRIDLibCtl.HCELL
        hFocusCell = .Items.ItemCell(.Items.FocusItem, .FocusColumnIndex)
        If (.SelectColumnInner > 0) Then
            ' Do we selected an inner cell?
            hFocusCell = .Items.InnerCell(, hFocusCell, .SelectColumnInner)
        End If
        Debug.Print "Focus = " & .Items.CellCaption(, hFocusCell) & " (" & .FocusColumnIndex
```

```
& "," & .SelectColumnInner & ")"
   End With
End Sub
```

The following C++ sample displays the focused cell:

```
#include "Items.h"
void OnFocusChangedGrid1()
{
   if ( IsWindow( m_grid.m_hWnd ) )
   {
      CItems items = m_grid.GetItems();
      COleVariant vtItem( items.GetFocusItem() ), vtColumn(
m_grid.GetFocusColumnIndex() );
      CString strFormat;
      strFormat.Format( "Focus on '%s'", V2S( &items.GetCellValue( vtItem, vtColumn ) ) );
      OutputDebugString( strFormat );
   }
}
```

The following C++ sample displays the focused cell, if the control contains inner cells:

```
#include "Items.h"
void OnFocusChangedGrid1()
{
   if ( IsWindow( m_grid.m_hWnd ) )
   {
      CItems items = m_grid.GetItems();
      COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
      COleVariant vtFocusCell( items.GetItemCell( items.GetFocusItem(),
COleVariant(m_grid.GetFocusColumnIndex())));
      if ( m_grid.GetSelectColumnInner() > 0 )
         vtFocusCell = items.GetInnerCell( vtMissing, vtFocusCell,
COleVariant(m_grid.GetSelectColumnInner()));
      CString strFormat;
      strFormat.Format( "Focus on '%s'", V2S( &items.GetCellValue( vtMissing, vtFocusCell )
) );
      OutputDebugString( strFormat );
```

```
    }
}
```

The following VB.NET sample displays the focused cell:

```
Private Sub AxGrid1_FocusChanged(ByVal sender As Object, ByVal e As System.EventArgs)
Handles AxGrid1.FocusChanged
    With AxGrid1.Items
        Debug.Print("Focus on '" & .CellValue(.FocusItem,
AxGrid1.FocusColumnIndex()).ToString() & "'")
    End With
End Sub
```

The following VB.NET sample displays the focused cell, if the control contains inner cells:

```
Private Sub AxGrid1_FocusChanged(ByVal sender As Object, ByVal e As System.EventArgs)
Handles AxGrid1.FocusChanged
    With AxGrid1.Items
        Dim focusCell As Object = .ItemCell(.FocusItem, AxGrid1.FocusColumnIndex)
        If (AxGrid1.SelectColumnInner > 0) Then
            focusCell = .InnerCell(Nothing, focusCell, AxGrid1.SelectColumnInner)
        End If
        Debug.Print("Focus on '" & .CellValue(Nothing, focusCell).ToString() & "'")
    End With
End Sub
```

The following C# sample displays the focused cell:

```
private void axGrid1_FocusChanged(object sender, EventArgs e)
{
    object focusValue = axGrid1.Items.get_CellValue(axGrid1.Items.FocusItem,
axGrid1.FocusColumnIndex);
    System.Diagnostics.Debug.WriteLine("Focus on '" + (focusValue != null ?
focusValue.ToString() : "" ) + "'");
}
```

The following C# sample displays the focused cell, if the control contains inner cells:

```
private void axGrid1_FocusChanged(object sender, EventArgs e)
```

```
{
   object focusCell = axGrid1.Items.get_ItemCell(axGrid1.Items.FocusItem,
axGrid1.FocusColumnIndex);
   if ( axGrid1.SelectColumnInner > 0 )
      focusCell = axGrid1.Items.get_InnerCell( null, focusCell, axGrid1.SelectColumnInner );
   object focusValue = axGrid1.Items.get_CellValue(null, focusCell);
   System.Diagnostics.Debug.WriteLine("Focus on '" + (focusValue != null ?
focusValue.ToString() : "" ) + "'");
}
```

The following VFP sample displays the focused cell:

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

with thisform.Grid1.Items
   .DefaultItem = .FocusItem
   wait window nowait .CellCaption( 0, thisform.Grid1.FocusColumnIndex() )
endwith
```

event FormatColumn (Item as HITEM, ColIndex as Long, Value as Variant)

Fired when a cell requires to format its value.

Type	Description
Item as HITEM	A long value that indicates the handle of the item being formatted
ColIndex as Long	A long expression that indicates the column's index being formatted
Value as Variant	A Variant value that should be converted.

Use the FormatColumn event to display a string different than the CellValue property. The FormatColumn event is fired only if the FireFormatColumn property of the Column is True. The FormatColumn event lets the user to provide the cell's caption before it is displayed on the control's list. For instance, the FormatColumn event is useful when the column cells contains prices(numbers), and you want to display that column formatted as currency, like $50 instead 50. Also, you can use the FormatColumn event to display item's index in the column, or to display the result of some operations based on the cells in the item (totals, currency conversion and so on). The FormatCell property indicates the individually predefined format to be applied to particular cells. The FormatColumn property applies the predefined format for all cells in the columns.

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.

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,
AxEXGRIDLib._IGridEvents_FormatColumnEvent e)
{
}
```

C++

```
void OnFormatColumn(long Item,long ColIndex,VARIANT FAR* Value)
{
}
```

C++ Builder

```
void __fastcall FormatColumn(TObject *Sender,Exgridlib_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_FormatColumnEvent) Handles FormatColumn
End Sub
```

VB6

```
Private Sub FormatColumn(ByVal Item As EXGRIDLibCtl.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(oGrid,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.Grid.1::HITEM,ColIndex as
N,Value as A)
end function
```

dBASE

```
function nativeObject_FormatColumn(Item,ColIndex,Value)
return
```

The following VB sample formats the second column to display the values using the currency format:

```
Private Sub Form_Load()
With Grid1
   .BeginUpdate
   .Columns.Add "A"
   .Columns.Add("B").FireFormatColumn = True ' Index of it is 1
   With .Items
      .AddItem Array("One", 1)
      .AddItem Array("Two", 2)
   End With
   .EndUpdate
End With
End Sub

Private Sub Grid1_FormatColumn(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As
Long, Value As Variant)
   Value = FormatCurrency(Value, 2, vbUseDefault)
End Sub
```

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.

```
Private Sub Grid1_FormatColumn(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As
Long, Value As Variant)
   On Error Resume Next
   With Grid1
      Value = FormatCurrency(Value)
   End With
End Sub
```

The following VB sample formats a column that contains date values. The FormatDateTime function is a VB function that returns an expression formatted as a date or time:

```
Private Sub Grid1_FormatColumn(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As
Long, Value As Variant)
   On Error Resume Next
```

```
    With Grid1
        Value = FormatDateTime(Value, vbLongDate)
    End With
End Sub
```

The following VB sample computes fields 1 + 2:

```
Private Sub Grid1_FormatColumn(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As
Long, Value As Variant)
    ' Adds the first two columns, or concaternates the strings
    On Error Resume Next
    With Grid1.Items
        Value = .CellValue(Item) + .CellValue(Item, 1)
    End With
End Sub
```

The following C++ sample displays a date column using a format like "Saturday, March 10, 2004":

```
void OnFormatColumnGrid1(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 AxGrid1_FormatColumn(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_FormatColumnEvent) Handles AxGrid1.FormatColumn
    e.value = DateTime.Parse(e.value).ToLongDateString()
End Sub
```

The following C# sample displays a date column using LongDate format:

```
private void axGrid1_FormatColumn(object sender,
AxEXGRIDLib._IGridEvents_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.Grid1.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 HITEM value that indicates the handle of the item being clicked.
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,
AxEXGRIDLib._IGridEvents_HyperLinkClickEvent e)
{
}
```

C++

```
void OnHyperLinkClick(long Item,long ColIndex)
{
}
```

C++ Builder

```
void __fastcall HyperLinkClick(TObject *Sender,Exgridlib_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_HyperLinkClickEvent) Handles HyperLinkClick
End Sub
```

VB6

```
Private Sub HyperLinkClick(ByVal Item As EXGRIDLibCtl.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(oGrid,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.Grid.1::HITEM,ColIndex as N)
end function
```

dBASE

```
function nativeObject_HyperLinkClick(Item,ColIndex)
return
```

The following VB sample displays the cell's value that's been clicked:

```
Private Sub Grid1_HyperLinkClick(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As
Long)
    ' Displays the cell's value that's been clicked
    Debug.Print Grid1.Items.CellValue(Item, ColIndex)
End Sub
```

The following C++ sample displays the caption of the hyperlink cell that's been clicked:

```
#include "Items.h"
void CYDlg::OnHyperLinkClickGrid1(long Item, long ColIndex)
{
    CItems items = m_grid.GetItems();
    COleVariant vtItem( Item ), vtColumn( ColIndex );
    OutputDebugString( V2S( &items.GetCellValue( vtItem, vtColumn ) ) );
}
```

The following VB.NET sample displays the caption of the hyperlink cell that's been clicked:

```
Private Sub AxGrid1_HyperLinkClick(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_HyperLinkClickEvent) Handles AxGrid1.HyperLinkClick
   With AxGrid1.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 axGrid1_HyperLinkClick(object sender,
AxEXGRIDLib._IGridEvents_HyperLinkClickEvent e)
{
   System.Diagnostics.Debug.WriteLine(axGrid1.Items.get_CellValue(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.Grid1.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	Specifies the handle of the item that contains the ActiveX control
Ev as OleEvent	A OleEvent object that contains information about the event.

The Exontrol's ExGrid 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.EXGRIDLib.OleEvent Ev)
{
}
```

VB

```
Private Sub ItemOleEvent(ByVal sender As System.Object,ByVal Item As
Integer,ByVal Ev As exontrol.EXGRIDLib.OleEvent) Handles ItemOleEvent
End Sub
```

Syntax for ItemOleEvent event, **/COM** version, on:

C#

```
private void ItemOleEvent(object sender,
AxEXGRIDLib._IGridEvents_ItemOleEventEvent e)
{
}
```

C++

```
void OnItemOleEvent(long Item,LPDISPATCH Ev)
{
}
```

C++ Builder

```
void __fastcall ItemOleEvent(TObject *Sender,Exgridlib_tlb::HITEM
Item,Exgridlib_tlb::IOleEvent *Ev)
{
```

```
}
```

Delphi

```
procedure ItemOleEvent(ASender: TObject; Item : HITEM;Ev : IOleEvent);
begin
end;
```

Delphi 8 (.NET only)

```
procedure ItemOleEvent(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_ItemOleEventEvent) Handles ItemOleEvent
End Sub
```

VB6

```
Private Sub ItemOleEvent(ByVal Item As EXGRIDLibCtl.HITEM,ByVal Ev As
EXGRIDLibCtl.IOleEvent)
End Sub
```

VBA

```
Private Sub ItemOleEvent(ByVal Item As Long,ByVal Ev As Object)
End Sub
```

VFP

```
LPARAMETERS Item,Ev
```

Xbas…

```
PROCEDURE OnItemOleEvent(oGrid,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)
{
}
```

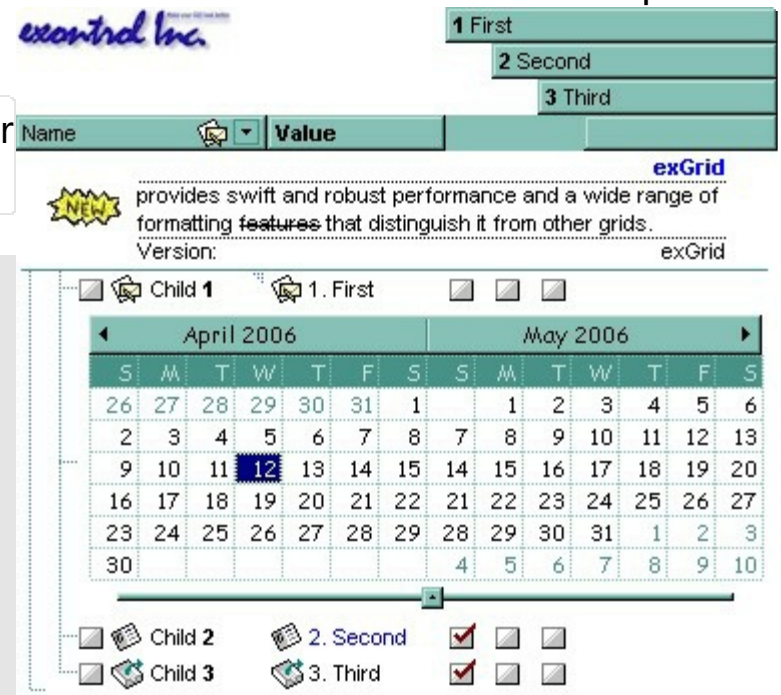
XBasic

```
function ItemOleEvent as v (Item as OLE::Exontrol.Grid.1::HITEM,Ev as
OLE::Exontrol.Grid.1::IOleEvent)
end function
```

dBASE

```
function nativeObject_ItemOleEvent(Item,Ev)
return
```

The following VB sample adds an item that hosts the Exontrol Calendar Control and prints each event fired by that ActiveX control:




```
Grid1.Items.ItemHeight(Grid1.Items.InsertContr
 "Exontrol.Calendar")) = 256
```

```
Private Sub Grid1_ItemOleEvent(ByVal Item As
EXGRIDLibCtl.HITEM, ByVal Ev As
EXGRIDLibCtl.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 VB6 sample shows you how to handle an event from a outer-inner-inner control. In other words, you have a master control (outer), which insert another control (inner), which insert another control (inner).

```
Private Sub expandItem(ByVal grid As Object, ByVal item As Long, ByVal level As
Long)
  Debug.Print "Expand item in " & level & " control"
End Sub

' BeforeExpandItem event - Fired before an item is about to be expanded (collapsed).
Private Sub Grid1_BeforeExpandItem(ByVal item As EXGRIDLibCtl.HITEM, Cancel As
Variant)
  expandItem Grid1.Object, item, 0
End Sub

' ItemOleEvent event - Fired when an ActiveX control hosted by an item has fired an event.
Private Sub Grid1_ItemOleEvent(ByVal item As EXGRIDLibCtl.HITEM, ByVal Ev As
EXGRIDLibCtl.IOleEvent)
  With Grid1
    'Debug.Print Ev.ToString()
    If (Ev.ID = 12) Then   ' BeforeExpandItem
      expandItem Grid1.Items.ItemObject(item), Ev.Param(0).Value, 1
    Else
      If (Ev.ID = 14) Then   ' ItemOLEEvent
        'Debug.Print Ev.Param(1).Value.ToString()
        If (Ev.Param(1).Value.ID = 12) Then    ' BeforeExpandItem
          'Debug.Print "Expand item in inner-inner control"
          expandItem
Grid1.Items.ItemObject(item).Items.ItemObject(Ev.Param(0).Value),
Ev.Param(1).Value.Param(0).Value, 2
        End If
```

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

This technique can be applied to ANY other event of the control, so you have a single function to be used when different events are fired.

The following C++ sample displays the events that an ActiveX control is firing while it is hosted by an item:

```
#import <exgrid.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 OnItemOleEventGrid1(long Item, LPDISPATCH Ev)
{
  EXGRIDLib::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++ )
```

```
        {
            EXGRIDLib::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 EXGRIDLib namespace that include all objects and types of the control's TypeLibrary. In case your exgrid.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 AxGrid1_ItemOleEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_ItemOleEventEvent) Handles AxGrid1.ItemOleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
    Dim i As Long
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXGRIDLib.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 axGrid1_ItemOleEvent(object sender,
AxEXGRIDLib._IGridEvents_ItemOleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine( "Event's name: " + e.ev.Name.ToString() );
    for ( int i= 0; i < e.ev.CountParam ; i++ )
```

```
    {
        EXGRIDLib.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 (KeyCode as Integer, Shift as Integer)

Occurs when the user presses a key while an object has the focus.

Type	Description
KeyCode as Integer	An integer that represent the key code
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

Use KeyDown and KeyUp event procedures if you need to respond to both the pressing and releasing of a key. You test for a condition by first assigning each result to a temporary integer variable and then comparing shift to a bit mask. Use the Change event to notify your application that the user changes the cell's value. Use the Editing property to determine whether the control is in edit mode. 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, AxEXGRIDLib._IGridEvents_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_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(oGrid,KeyCode,Shift)
RETURN
```

Syntax for KeyDown event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="KeyDown(KeyCode,Shift)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function KeyDown(KeyCode,Shift)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComKeyDown Short llKeyCode Short llShift
   Forward Send OnComKeyDown llKeyCode llShift
End_Procedure
```

Visual Objects

```
METHOD OCX_KeyDown(KeyCode,Shift) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_KeyDown(COMVariant /*short*/ _KeyCode,int _Shift)
{
}
```

XBasic

```
function KeyDown as v (KeyCode as N,Shift as N)
end function
```

dBASE

```
function nativeObject_KeyDown(KeyCode,Shift)
return
```

The following VB sample starts editing the cell is the user presses the F4 key, and AutoEdit property is False.

```
Private Sub Grid1_KeyDown(KeyCode As Integer, Shift As Integer)
   ' Edits the focused cell once that user presses the F4 key
   If (KeyCode = vbKeyF4) Then
```

```
    Grid1.Edit
  End If
End Sub
```

The following VB sample advances to the next field, when the user presses the ENTER key (the sample is useful when the current editor is a simple edit control) :

```
Private Sub Grid1_KeyDown(KeyCode As Integer, Shift As Integer)
  If (KeyCode = vbKeyReturn) Then
    KeyCode = vbKeyDown
  End If
End Sub
```

The following C++ sample starts editing the focused cell when user presses the ENTER key, (AutoEdit property is False):

```
void OnKeyDownGrid1(short FAR* KeyCode, short Shift)
{
  if ( *KeyCode == VK_RETURN )
  {
    COleVariant vtMissing; V_VT( &vtMissing) = VT_ERROR;
    if ( m_grid.GetEditing() == 0 )
      m_grid.Edit( vtMissing );
  }
}
```

The following VB.NET sample starts editing the focused cell when user presses the ENTER key, (AutoEdit property is False):

```
Private Sub AxGrid1_KeyDownEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_KeyDownEvent) Handles AxGrid1.KeyDownEvent
  If (Convert.ToUInt32(e.keyCode) = Convert.ToUInt32(Keys.Enter)) Then
    AxGrid1.Edit(Nothing)
  End If
End Sub
```

The following C# sample starts editing the focused cell when user presses the ENTER key, (AutoEdit property is False):

```
private void axGrid1_KeyDownEvent(object sender,
```

```
AxEXGRIDLib._IGridEvents_KeyDownEvent e)
{
    if (Convert.ToUInt32(e.keyCode) == Convert.ToUInt32(Keys.Enter))
        axGrid1.Edit(null);
}
```

The following VFP sample starts editing the focused cell when user presses the ENTER key, (AutoEdit property is False):

```
*** ActiveX Control Event ***
LPARAMETERS keycode, shift

if ( keycode = 13 )
    thisform.Grid1.Object.Edit("")
endif
```

event KeyPress (KeyAscii as Integer)

Occurs when the user presses and releases an ANSI key.

Type	Description
KeyAscii as Integer	An integer that returns a standard numeric ANSI keycode

The KeyPress event lets you immediately test keystrokes for validity or for formatting characters as they are typed. Changing the value of the keyascii argument changes the character displayed. Use KeyDown and KeyUp event procedures to handle any keystroke not recognized by KeyPress, such as function keys, editing keys, navigation keys, and any combinations of these with keyboard modifiers. Unlike the KeyDown and KeyUp events, KeyPress does not indicate the physical state of the keyboard; instead, it passes a character. KeyPress interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters. Use the Change event to notify your application that the user changes the cell's value. Use the Editing property to determine whether the control is in edit mode.

Syntax for KeyPress event, **/NET** version, on:

C#

```
private void KeyPress(object sender,ref short KeyAscii)
{
}
```

VB

```
Private Sub KeyPress(ByVal sender As System.Object,ByRef KeyAscii As Short)
Handles KeyPress
End Sub
```

Syntax for KeyPress event, **/COM** version, on:

C#

```
private void KeyPressEvent(object sender,
AxEXGRIDLib._IGridEvents_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_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(oGrid,KeyAscii)
RETURN
```

Syntax for KeyPress event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="KeyPress(KeyAscii)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function KeyPress(KeyAscii)
End Function
```

```
</SCRIPT>
```

Visual Data…

```
Procedure OnComKeyPress Short llKeyAscii
    Forward Send OnComKeyPress llKeyAscii
End_Procedure
```

Visual Objects

```
METHOD OCX_KeyPress(KeyAscii) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_KeyPress(COMVariant /*short*/ _KeyAscii)
{
}
```

XBasic

```
function KeyPress as v (KeyAscii as N)
end function
```

dBASE

```
function nativeObject_KeyPress(KeyAscii)
return
```

event KeyUp (KeyCode as Integer, Shift as Integer)

Occurs when the user releases a key while an object has the focus.

Type	Description
KeyCode as Integer	An integer that represent the key code.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

Use the KeyUp event procedure to respond to the releasing of a key. Use the Change event to notify your application that the user changes the cell's value. Use the Editing property to determine whether the control is in edit mode.

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, AxEXGRIDLib._IGridEvents_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_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(oGrid,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 notifies your application once a column is resized or moved by drag and drop. Also, the LayoutChanged event may be fired if the item's position is changed by drag and drop using the AutoDrag property.

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(oGrid)
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
```

Since, the LayotChanged event may be fired on different scenarios, you can distingue the action that previously occurs by storing the ItemFromPoint and/or ColumnFromPoint during the MouseDown event like in the following VB sample:

```
Dim iItemFromPointMouseDown As Long
Dim iColumnFromPointMouseDown As Long

Private Sub Form_Load()
   iItemFromPointMouseDown = 0
   iColumnFromPointMouseDown = -1
End Sub

Private Sub Grid1_LayoutChanged()
   If (iItemFromPointMouseDown <> 0) Then
         Debug.Print "Items section changed"
      Else
      If (iColumnFromPointMouseDown <> -1) Then
         Debug.Print "Columns section changed"
      Else
         Debug.Print "Others"
      End If
   End If
   iItemFromPointMouseDown = 0
   iColumnFromPointMouseDown = -1
End Sub
```

```
Private Sub Grid1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
   Dim c As Long, hit As HitTestInfoEnum
   With Grid1
      iItemFromPointMouseDown = .ItemFromPoint(-1, -1, c, hit)
      iColumnFromPointMouseDown = .ColumnFromPoint(-1, -1)
   End With
End Sub
```

The sample displays:

- "Columns section changed" if any change occurs in the Columns section, like moving a column to a new position or resizing the column.
- "Items section changed", if the user drags an item to a new position using the AutoDrag property on exAutoDragPosition, exAutoDragPositionKeepIndent and exAutoDragPositionAny

You can 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.

There are two options to avoid losing the columns proportions:

- Avoiding resizing the control under a specified width, like in the sample:

```
Private Sub Form_Resize()
On Error Resume Next
   If ScaleWidth / Screen.TwipsPerPixelX > 64 Then
      With Grid1
         .Left = 0
         .Top = 0
         .Width = ScaleWidth
         .Height = ScaleHeight
      End With
   End If
End Sub
```

- Using the LayoutChanged event to store the columns proportions manually. The

following sample holds the columns proportions when LayoutChanged event is fired. The sample ensures that the proportions are saved only when the user resizes on of the control's columns, not when the user resizes the entire control. The proportions are kept by the Data property of the Column object. The sample can be changed smoothly by using a simple collection to hold the columns proportions instead using the Data property of the Column object

```
Option Explicit
Dim nFit As Long
Private Declare Function PeekMessage Lib "user32" Alias "PeekMessageA"
(lpMsg As MSG, ByVal hwnd As Long, ByVal wMsgFilterMin As Long, ByVal
wMsgFilterMax As Long, ByVal wRemoveMsg As Long) As Long
Private Declare Function TranslateMessage Lib "user32" (lpMsg As MSG) As Long
Private Declare Function DispatchMessage Lib "user32" Alias
"DispatchMessageA" (lpMsg As MSG) As Long
Private Const PM_REMOVE = &amp;amp;H1
Private Type POINTAPI
      x As Long
      y As Long
End Type
Private Type MSG
   hwnd As Long
   message As Long
   wParam As Long
   lParam As Long
   time As Long
   pt As POINTAPI
End Type

Private Sub Form_Load()
   nFit = 0

   onGridResize Grid1
End Sub

Private Sub Form_Resize()
On Error Resume Next
   nFit = nFit + 1
```

```
    With Grid1
        .Left = 0
        .Top = 0
        .Width = ScaleWidth
        .Height = ScaleHeight
    End With
    fit Grid1

    nFit = nFit - 1
End Sub

Private Sub Grid1_LayoutChanged()
    If (nFit = 0) Then
        onGridResize Grid1
    End If
End Sub

Private Sub fit(ByVal g As EXGRIDLibCtl.Grid)
    nFit = nFit + 1
    With g
        If (.ColumnAutoResize) Then
            .BeginUpdate
            .ColumnAutoResize = False
            Dim c As EXGRIDLibCtl.Column
            For Each c In .Columns
                c.Width = c.Data
            Next
            .ColumnAutoResize = True
            .EndUpdate
        End If
    End With
    waitToProcessMessages
    nFit = nFit - 1
End Sub

Private Sub onGridResize(ByVal g As EXGRIDLibCtl.Grid)
    Dim c As Object
```

```
    With g
        If (.ColumnAutoResize) Then
            For Each c In .Columns
                c.Data = c.Width
            Next
        End If
    End With
End Sub

Private Sub waitToProcessMessages()
    Dim m As MSG
    While PeekMessage(m, 0, 0, 0, PM_REMOVE)
        TranslateMessage m
        DispatchMessage m
    Wend
End Sub
```

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. Use the AnchorFromPoint property to retrieve the identifier of the anchor element from the point. The AnchorClick event notifies your application that the user clicks an <a> anchor element.

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,
```

```
AxEXGRIDLib._IGridEvents_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_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(oGrid,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.Grid.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Grid.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's been clicked:

```
Private Sub Grid1_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 EXGRIDLibCtl.HitTestInfoEnum
    ' Gets the item from (X,Y)
    h = Grid1.ItemFromPoint(X, Y, c, hit)
    If Not (h = 0) Then
        Debug.Print Grid1.Items.CellValue(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 Grid1_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 = Grid1.ItemFromPoint(X, Y, c, hit)
        If Not (h = 0) Then
            Dim i As Long
            PopupMenu1.Items.Add Grid1.Items.CellValue(h, c)
            i = PopupMenu1.ShowAtCursor
        End If
    End If
End Sub
```

The following C++ 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 OnMouseDownGrid1(short Button, short Shift, long X, long Y)
{
    long c = 0, hit = 0, hItem = m_grid.GetItemFromPoint( X, Y, &c, &hit );
    if ( ( hItem != 0 ) || ( c != 0 ) )
    {
        CItems items = m_grid.GetItems();
        COleVariant vtItem( hItem ), vtColumn( c );
        CString strCaption = V2S( &items.GetCellValue( 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 AxGrid1_MouseDownEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_MouseDownEvent) Handles AxGrid1.MouseDownEvent
    With AxGrid1
        Dim i As Integer, c As Integer, hit As EXGRIDLib.HitTestInfoEnum
        i = .get_ItemFromPoint(e.x, e.y, c, hit)
        If (Not (i = 0) Or Not (c = 0)) Then
            Debug.WriteLine("Cell: " & .Items.CellValue(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 axGrid1_MouseDownEvent(object sender,
AxEXGRIDLib._IGridEvents_MouseDownEvent e)
{
   int c = 0;
   EXGRIDLib.HitTestInfoEnum hit;
   int i = axGrid1.get_ItemFromPoint( e.x, e.y, out c,out hit );
   if ( ( i != 0 ) || ( c != 0 ) )
   {
      string s = axGrid1.Items.get_CellValue( 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.Grid1
   .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
   if ( .Items.DefaultItem <> 0 ) or ( c <> 0 )
      wait window nowait .Items.CellValue( 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.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates

The MouseMove event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseMove event whenever the mouse position is within its borders. Use the ItemFromPoint property to get the item from cursor. Use the ColumnFromPoint property to get the column from point. The Background(exCursorHoverColumn) property specifies the visual appearance of the column's header when the cursor hovers it. Use the AnchorFromPoint property to retrieve the identifier of the anchor element from the point. The WordFromPoint property determines the word from the cursor.

Syntax for MouseMove event, **/NET** version, on:

C#
```
private void MouseMoveEvent(object sender,short Button,short Shift,int X,int Y)
{
}
```

VB
```
Private Sub MouseMoveEvent(ByVal sender As System.Object,ByVal Button As
Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles
MouseMoveEvent
End Sub
```

Syntax for MouseMove event, **/COM** version, on:

C#
```
private void MouseMoveEvent(object sender,
AxEXGRIDLib._IGridEvents_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_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(oGrid,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.Grid.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Grid.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 Grid1_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 EXGRIDLibCtl.HitTestInfoEnum
   ' Gets the item from (X,Y)
   h = Grid1.ItemFromPoint(X, Y, c, hit)
   If Not (h = 0) Then
      Debug.Print Grid1.Items.CellValue(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 Grid1_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 EXGRIDLibCtl.HitTestInfoEnum
   ' Gets the item from (X,Y)
   h = Grid1.ItemFromPoint(X, Y, c, hit)
   If Not (h = 0) Or Not (c = 0) Then
      Debug.Print Grid1.Items.CellValue(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 OnMouseMoveGrid1(short Button, short Shift, long X, long Y)
{
   long c = 0, hit = 0, hItem = m_grid.GetItemFromPoint( X, Y, &c, &hit );
   if ( ( hItem != 0 ) || ( c != 0 ) )
   {
      CItems items = m_grid.GetItems();
      COleVariant vtItem( hItem ), vtColumn( c );
      CString strCaption = V2S( &items.GetCellValue( 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 AxGrid1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_MouseMoveEvent) Handles AxGrid1.MouseMoveEvent
   With AxGrid1
      Dim i As Integer, c As Integer, hit As EXGRIDLib.HitTestInfoEnum
      i = .get_ItemFromPoint(e.x, e.y, c, hit)
      If (Not (i = 0) Or Not (c = 0)) Then
         Debug.WriteLine("Cell: " & .Items.CellValue(i, c) & " Hit: " & hit.ToString())
```

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

The following C# sample displays the cell's from the point:

```
private void axGrid1_MouseMoveEvent(object sender,
AxEXGRIDLib._IGridEvents_MouseMoveEvent e)
{
    int c = 0;
    EXGRIDLib.HitTestInfoEnum hit;
    int i = axGrid1.get_ItemFromPoint( e.x, e.y, out c,out hit );
    if ( ( i != 0 ) || ( c != 0 ) )
    {
        object cap = axGrid1.Items.get_CellValue(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.Grid1
    .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
    if ( .Items.DefaultItem <> 0 ) or ( c <> 0 )
        wait window nowait .Items.CellValue( 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. Use the AnchorFromPoint property to retrieve the identifier of the anchor element from the point. The AnchorClick event notifies your application that the user clicks an <a> anchor element.

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,
```

```
AxEXGRIDLib._IGridEvents_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_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(oGrid,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.Grid.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Grid.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 Grid1_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 = Grid1.ItemFromPoint(X, Y, c, hit)
    If Not (h = 0) Then
        Debug.Print Grid1.Items.CellValue(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 Grid1_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 = Grid1.ItemFromPoint(X, Y, c, hit)
        If Not (h = 0) Then
            Dim i As Long
            PopupMenu1.Items.Add Grid1.Items.CellValue(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 OnMouseUpGrid1(short Button, short Shift, long X, long Y)
{
    long c = 0, hit = 0, hItem = m_grid.GetItemFromPoint( X, Y, &c, &hit );
    if ( ( hItem != 0 ) || ( c != 0 ) )
    {
        CItems items = m_grid.GetItems();
        COleVariant vtItem( hItem ), vtColumn( c );
        CString strCaption = V2S( &items.GetCellValue( 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 AxGrid1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_MouseUpEvent) Handles AxGrid1.MouseUpEvent
    With AxGrid1
        Dim i As Integer, c As Integer, hit As EXGRIDLib.HitTestInfoEnum
        i = .get_ItemFromPoint(e.x, e.y, c, hit)
        If (Not (i = 0) Or Not (c = 0)) Then
            Debug.WriteLine("Cell: " & .Items.CellValue(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 axGrid1_MouseUpEvent(object sender,
AxEXGRIDLib._IGridEvents_MouseUpEvent e)
{
   int c = 0;
   EXGRIDLib.HitTestInfoEnum hit;
   int i = axGrid1.get_ItemFromPoint( e.x, e.y, out c,out hit );
   if ( ( i != 0 ) || ( c != 0 ) )
   {
      string s = axGrid1.Items.get_CellValue( 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.Grid1
   .Items.DefaultItem = .ItemFromPoint( x, y, @c, @hit )
   if ( .Items.DefaultItem <> 0 ) or ( c <> 0 )
      wait window nowait .Items.CellValue( 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.

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, AxEXGRIDLib._IGridEvents_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_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(oGrid,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 Grid1_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 OnOffsetChangedGrid1(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 AxGrid1_OffsetChanged(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_OffsetChangedEvent) Handles AxGrid1.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 axGrid1_OffsetChanged(object sender,
AxEXGRIDLib._IGridEvents_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,
AxEXGRIDLib._IGridEvents_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_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(oGrid,Effect)
RETURN
```

Syntax for OLECompleteDrag event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="OLECompleteDrag(Effect)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function OLECompleteDrag(Effect)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComOLECompleteDrag Integer llEffect
   Forward Send OnComOLECompleteDrag llEffect
End_Procedure
```

Visual Objects

```
METHOD OCX_OLECompleteDrag(Effect) CLASS MainDialog
RETURN NIL
```

X++

```
// OLECompleteDrag event is not supported. Use the
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

XBasic

```
function OLECompleteDrag as v (Effect as N)
end function
```

dBASE

```
function nativeObject_OLECompleteDrag(Effect)
return
```

event OLEDragDrop (Data as ExDataObject, Effect as Long, Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when a source component is dropped onto a target component when the source component determines that a drop can occur.

Type	Description
Data as ExDataObject	An ExDataObject object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the ExDataObject, it is provided when the control calls the GetData method. The SetData and Clear methods cannot be used here.
Effect as Long	A Long set by the target component identifying the action that has been performed (if any), thus allowing the source to take appropriate action if the component was moved (such as the source deleting the data). The possible values are listed in bellow.
Button as Integer	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are depressed
Shift as Integer	An integer which acts as a bit field corresponding to the state of the SHIFT, CTRL, and ALT keys when they are depressed. The SHIFT key is bit 0, the CTRL key is bit 1, and the ALT key is bit 2. These bits correspond to the values 1, 2, and 4, respectively. The shift parameter indicates the state of these keys; some, all, or none of the bits can be set, indicating that some, all, or none of the keys are depressed. For example, if both the CTRL and ALT keys were depressed, the value of shift would be 6.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The X value is always expressed in container coordinates
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The Y value is always expressed in container coordinates.

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,
AxEXGRIDLib._IGridEvents_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,Exgridlib_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_OLEDragDropEvent) Handles OLEDragDrop
End Sub
```

VB6

```
Private Sub OLEDragDrop(ByVal Data As EXGRIDLibCtl.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(oGrid,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.Grid.1::IExDataObject,Effect as
N,Button as N,Shift as N,X as OLE::Exontrol.Grid.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Grid.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 Grid1_OLEDragDrop(Index As Integer, ByVal Data As
EXGRIDLibCtl.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
      Grid1(Index).Visible = False
      With Grid1(Index)
      .BeginUpdate
         Dim i As HITEM
         i = .Items.AddItem(strFile)
         .Items.EnsureVisibleItem i
      .EndUpdate
      End With
      Grid1(Index).Visible = True
   End If
End Sub
```

The following VC sample inserts a child item for each file that user drags:

```
#import <exgrid.dll> rename( "GetItems", "exGetItems" )

#include "Items.h"
void OnOLEDragDropGrid1(LPDISPATCH Data, long FAR* Effect, short Button, short Shift,
long X, long Y)
{
   EXGRIDLib::IExDataObjectPtr spData( Data );
   if ( spData != NULL )
      if ( spData->GetFormat( EXGRIDLib::exCFFiles ) )
      {
         CItems items = m_grid.GetItems();
         // Gets the handle of the item where the files will be inserted
         long c = 0, h = 0, nParentItem = m_grid.GetItemFromPoint( X, Y, &c, &h );
         if ( nParentItem == 0 )
            if ( c != 0 )
               nParentItem = items.GetCellItem( c );
         EXGRIDLib::IExDataObjectFilesPtr spFiles( spData->Files );
```

```
        if ( spFiles->Count > 0 )
        {
            m_grid.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_grid.EndUpdate();
        }

    }
}
```

The #import statement imports definition for the ExDataObject and ExDataObjectFiles objects. If the exgrid.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 AxGrid1_OLEDragDrop(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_OLEDragDropEvent) Handles AxGrid1.OLEDragDrop
    If e.data.GetFormat(EXGRIDLib.exClipboardFormatEnum.exCFFiles) Then
        If (e.data.Files.Count > 0) Then
            AxGrid1.BeginUpdate()
            With AxGrid1.Items
                Dim iParent As Integer, c As Integer, hit As EXGRIDLib.HitTestInfoEnum
                iParent = AxGrid1.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
        AxGrid1.EndUpdate()
    End If
  End If
End Sub
```

The following C# sample inserts a child item for each file that user drags:

```
private void axGrid1_OLEDragDrop(object sender,
AxEXGRIDLib._IGridEvents_OLEDragDropEvent e)
{
  if ( e.data.GetFormat( Convert.ToInt16(EXGRIDLib.exClipboardFormatEnum.exCFFiles) ) )
    if ( e.data.Files.Count > 0 )
    {
      EXGRIDLib.HitTestInfoEnum hit;
      int c = 0, iParent = axGrid1.get_ItemFromPoint( e.x, e.y, out c, out hit );
      if ( iParent == 0 )
        if ( c != 0 )
          iParent = axGrid1.Items.get_CellItem( c );

      axGrid1.BeginUpdate();
      for ( int i = 0; i < e.data.Files.Count; i++ )
        axGrid1.Items.InsertItem( iParent,"", e.data.Files[i].ToString() );
      if ( iParent != 0 )
        axGrid1.Items.set_ExpandItem( iParent, true );
      axGrid1.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.Grid1.Items
         iParent = thisform.Grid1.ItemFromPoint( x, y, @c, @hit )

         thisform.Grid1.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.Grid1.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 bellow.
Button as Integer	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are depressed.
Shift as Integer	These bits correspond to the values 1, 2, and 4, respectively. The shift parameter indicates the state of these keys; some, all, or none of the bits can be set, indicating that some, all, or none of the keys are depressed. For example, if both the CTRL and ALT keys were depressed, the value of shift would be 6.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The X value is always expressed in container coordinates
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The Y value is always expressed in container coordinates.
State as Integer	An integer that corresponds to the transition state of the control being dragged in relation to a target form or control. The possible values are listed bellow.

The settings for effect are:

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

The settings for state are:

- exOLEDragEnter (0), Source component is being dragged within the range of a target.
- exOLEDragLeave (1), Source component is being dragged out of the range of a target.
- exOLEOLEDragOver (2), Source component has moved from one position in the target to another.

Note If the state parameter is 1, indicating that the mouse pointer has left the target, then the x and y parameters will contain zeros.
The source component should always mask values from the effect parameter to ensure compatibility with future implementations of ActiveX components. As a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.
For example, a source component should not compare an effect against, say, exOLEDropEffectCopy, such as in this manner:

If Effect = exOLEDropEffectCopy...
Instead, the source component should mask for the value or values being sought, such as this:
If Effect And exOLEDropEffectCopy = exOLEDropEffectCopy...
-or-
If (Effect And exOLEDropEffectCopy)...
This allows for the definition of new drop effects in future versions while preserving backwards compatibility with your existing code.
The control supports only manual OLE drag and drop events.

Syntax for OLEDragOver event, **/NET** version, on:

C#
```
// OLEDragOver event is not supported. Use the DragEnter,DragLeave,DragOver, DragDrop ... events.
```

VB
```
// OLEDragOver event is not supported. Use the DragEnter,DragLeave,DragOver, DragDrop ... events.
```

Syntax for OLEDragOver event, **/COM** version, on:

C#

```
private void OLEDragOver(object sender,
AxEXGRIDLib._IGridEvents_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,Exgridlib_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_OLEDragOverEvent) Handles OLEDragOver
End Sub
```

VB6

```
Private Sub OLEDragOver(ByVal Data As EXGRIDLibCtl.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(oGrid,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.Grid.1::IExDataObject,Effect as N,Button as N,Shift as N,X as OLE::Exontrol.Grid.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.Grid.1::OLE_YPOS_PIXELS,State as N)
```

```
end function
```

dBASE

```
function nativeObject_OLEDragOver(Data,Effect,Button,Shift,X,Y,State)
return
```

event OLEGiveFeedback (Effect as Long, DefaultCursors as Boolean)

Allows the drag source to specify the type of OLE drag-and-drop operation and the visual feedback.

Type	Description
Effect as Long	A long integer set by the target component in the OLEDragOver event specifying the action to be performed if the user drops the selection on it. This allows the source to take the appropriate action (such as giving visual feedback). The possible values are listed bellow.
DefaultCursors as Boolean	Boolean value that determines whether to use the default mouse cursor, or to use a user-defined mouse cursor.True (default) = use default mouse cursor.False = do not use default cursor. Mouse cursor must be set with the MousePointer property of the Screen object

The settings for Effect are:

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

If there is no code in the OLEGiveFeedback event, or if the defaultcursors parameter is set to True, the mouse cursor will be set to the default cursor provided by the control. The source component should always mask values from the effect parameter to ensure compatibility with future implementations of ActiveX components. As a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an effect against, say, exOLEDropEffectCopy, such as in this manner:
If Effect = exOLEDropEffectCopy...
Instead, the source component should mask for the value or values being sought, such as this:
If Effect And exOLEDropEffectCopy = exOLEDropEffectCopy...
-or-
If (Effect And exOLEDropEffectCopy)...
This allows for the definition of new drop effects in future versions while preserving backwards compatibility with your existing code.

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,
AxEXGRIDLib._IGridEvents_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_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(oGrid,Effect,DefaultCursors)
RETURN
```

Syntax for OLEGiveFeedback event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="OLEGiveFeedback(Effect,DefaultCursors)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function OLEGiveFeedback(Effect,DefaultCursors)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComOLEGiveFeedback Integer llEffect Boolean llDefaultCursors
    Forward Send OnComOLEGiveFeedback llEffect llDefaultCursors
End_Procedure
```

Visual Objects

```
METHOD OCX_OLEGiveFeedback(Effect,DefaultCursors) CLASS MainDialog
RETURN NIL
```

X++

```
// OLEGiveFeedback event is not supported. Use the
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

XBasic

```
function OLEGiveFeedback as v (Effect as N,DefaultCursors as L)
end function
```

dBASE

```
function nativeObject_OLEGiveFeedback(Effect,DefaultCursors)
return
```

event OLESetData (Data as ExDataObject, Format as Integer)

Occurs on a drag source when a drop target calls the GetData method and there is no data in a specified format in the OLE drag-and-drop DataObject.

Type	Description
Data as ExDataObject	An ExDataObject object in which to place the requested data. The component calls the SetData method to load the requested format.
Format as Integer	An integer specifying the format of the data that the target component is requesting. The source component uses this value to determine what to load into the ExDataObject object.

The OLESetData is not implemented

Syntax for OLESetData event, **/NET** version, on:

C#
```
// OLESetData event is not supported. Use the DragEnter,DragLeave,DragOver,
DragDrop ... events.
```

VB
```
// OLESetData event is not supported. Use the DragEnter,DragLeave,DragOver,
DragDrop ... events.
```

Syntax for OLESetData event, **/COM** version, on:

C#
```
private void OLESetData(object sender,
AxEXGRIDLib._IGridEvents_OLESetDataEvent e)
{
}
```

C++
```
void OnOLESetData(LPDISPATCH Data,short Format)
{
}
```

C++ Builder
```
void __fastcall OLESetData(TObject *Sender,Exgridlib_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_OLESetDataEvent) Handles OLESetData
End Sub
```

VB6

```
Private Sub OLESetData(ByVal Data As EXGRIDLibCtl.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(oGrid,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.Grid.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*

The settings for AllowEffects are:

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

The source component should logically Or together the supported values and places the result in the AllowedEffects parameter. The target component can use this value to determine the appropriate action (and what the appropriate user feedback should be). You may wish to defer putting data into the ExDataObject object until the target component requests it. This allows the source component to save time. If the user does not load any formats into the ExDataObject, then the drag/drop operation is canceled. Use exCFFiles and Files property to add files to the drag and drop data object.

The idea of drag and drop in exGrid control is the same as in other controls. To start accepting drag and drop sources the exGrid control should have the OLEDropMode to exOLEDropManual. Once that is is set, the exGrid starts accepting any drag and drop sources.

The first step is if you want to be able to drag items from your exGrid control to other

controls the idea is to handle the OLE_StartDrag event. The event passes an object ExDataObject (Data) as argument. The Data and AllowedEffects can be changed only in the OLEStartDrag event. The OLE_StartDrag event is fired when user is about to drag items from the control. **The AllowedEffect parameter and SetData property must be set to continue drag and drop operation, as in the following samples**:

Syntax for OLEStartDrag event, **/NET** version, on:

C#
```
// OLEStartDrag event is not supported. Use the DragEnter,DragLeave,DragOver, DragDrop ... events.
```

VB
```
// OLEStartDrag event is not supported. Use the DragEnter,DragLeave,DragOver, DragDrop ... events.
```

Syntax for OLEStartDrag event, **/COM** version, on:

C#
```
private void OLEStartDrag(object sender, AxEXGRIDLib._IGridEvents_OLEStartDragEvent e)
{
}
```

C++
```
void OnOLEStartDrag(LPDISPATCH Data,long FAR* AllowedEffects)
{
}
```

C++ Builder
```
void __fastcall OLEStartDrag(TObject *Sender,Exgridlib_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: AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_OLEStartDragEvent) Handles OLEStartDrag
End Sub
```

VB6

```
Private Sub OLEStartDrag(ByVal Data As
EXGRIDLibCtl.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(oGrid,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.Grid.1::IExDataObject,AllowedEffects as N)
end function
```

dBASE

```
function nativeObject_OLEStartDrag(Data,AllowedEffects)
return
```

The following VB sample drags data from a control to another, by registering a new clipboard format:

```
Private Sub Grid1_OLEStartDrag(Index As Integer, ByVal Data As
EXGRIDLibCtl.IExDataObject, AllowedEffects As Long)

  ' We are going to add two clipboard formats: text and "EXGRID" 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 Grid1(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 (Grid1(Index).Items.ItemParent(Grid1(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("EXGRID")

End Sub
```

The code fills data for two types of clipboard formats: text (CF_TEXT) and "EXGRID" 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 Grid1_OLEDragDrop(Index As Integer, ByVal Data As
EXGRIDLibCtl.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("EXGRID"))) 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("EXGRID"))
        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
        Grid1(Index).Visible = False
        With Grid1(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
        Grid1(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 <exgrid.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 OnOLEStartDragGrid1(LPDISPATCH Data, long FAR* AllowedEffects)
{
    CItems items = m_grid.GetItems();
    long nCount = items.GetSelectCount(), nColumnCount =
m_grid.GetColumns().GetCount();
    if ( nCount > 0 )
    {
        *AllowedEffects = /*exOLEDropEffectCopy */ 1;
        EXGRIDLib::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)EXGRIDLib::exCFText) );
        }
    }
}
```

The sample saves data as CF_TEXT format (EXGRIDLib::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 exgrid.dll file is located in another place than your system folder, the path to the exgrid.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 AxGrid1_OLEStartDrag(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_OLEStartDragEvent) Handles AxGrid1.OLEStartDrag
   With AxGrid1.Items
      If (.SelectCount > 0) Then
         e.allowedEffects = 1 'exOLEDropEffectCopy
         Dim i As Integer, j As Integer, strData As String, nColumnCount As Long =
AxGrid1.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, EXGRIDLib.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 axGrid1_OLEStartDrag(object sender,
AxEXGRIDLib._IGridEvents_OLEStartDragEvent e)
{
   int nCount = axGrid1.Items.SelectCount;
   if ( nCount > 0 )
   {
      int nColumnCount = axGrid1.Columns.Count;
      e.allowedEffects = /*exOLEDropEffectCopy*/ 1;
      string strData = "";
      for ( int i =0 ; i < nCount; i++ )
```

```
        {
            for ( int j = 0; j < nColumnCount; j++ )
            {
                object strCell = axGrid1.Items.get_CellCaption(axGrid1.Items.get_SelectedItem(i),
j);
                strData +=  ( strCell != null ? strCell.ToString() : "" ) + "\t";
            }
            strData += "\r\n";
        }
        e.data.SetData( strData, EXGRIDLib.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.Grid1.Items
    if ( .SelectCount() > 0 )
        allowedeffects = 1 && exOLEDropEffectCopy
        sData = ""
        nColumnCount = thisform.Grid1.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.

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, AxEXGRIDLib._IGridEvents_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_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(oGrid,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
```

Here's a trick that counts the number of items being shown while control is running in virtual mode (VirtualMode property is True):

```
Private Sub Grid1_OversizeChanged(ByVal Horizontal As Boolean, ByVal NewVal As Long)
    If Not Horizontal Then
        Debug.Print NewVal + Grid1.Items.VisibleCount
    End If
End Sub
```

The idea is to add the NewVal parameter of the OversizeChanged event, with VisibleCount property of the Items collection.

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(oGrid)
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 Grid1_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 Grid1_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 = Grid1.ItemFromPoint(X, Y, c, hit)
      If Not (h = 0) Then
         Dim i As Long
         PopupMenu1.Items.Add Grid1.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 OnMouseUpGrid1(short Button, short Shift, long X, long Y)
{
   long c = 0, hit = 0, hItem = m_grid.GetItemFromPoint( X, Y, &c, &hit );
   if ( ( hItem != 0 ) || ( c != 0 ) )
   {
      CItems items = m_grid.GetItems();
      COleVariant vtItem( hItem ), vtColumn( c );
      CString strCaption = V2S( &items.GetCellValue( 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 AxGrid1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_MouseUpEvent) Handles AxGrid1.MouseUpEvent
   With AxGrid1
      Dim i As Integer, c As Integer, hit As EXGRIDLib.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 axGrid1_MouseUpEvent(object sender,
AxEXGRIDLib._IGridEvents_MouseUpEvent e)
{
   int c = 0;
   EXGRIDLib.HitTestInfoEnum hit;
   int i = axGrid1.get_ItemFromPoint( e.x, e.y, out c,out hit );
   if ( ( i != 0 ) || ( c != 0 ) )
   {
      string s = axGrid1.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.Grid1
   .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 that is removing from the Columns collection.

The RemoveColumn event is invoked when the control is about to remove a column. Use the RemoveColumn event to release any extra data associated to the column. Use the Remove method to remove a specific column from Columns collection. Use the Clear method to clear the columns collection. Use the 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 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(oGrid)
RETURN
```

Syntax for RClick event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="RClick()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function RClick()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComRClick
   Forward Send OnComRClick
End_Procedure
```

Visual Objects

```
METHOD OCX_RClick() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_RClick()
{
}
```

XBasic

```
function RClick as v ()
end function
```

dBASE

```
function nativeObject_RClick()
return
```

event RemoveItem (Item as HITEM)

Occurs before deleting an Item.

Type	Description
Item as HITEM	A long expression that indicates the handle of item being deleted.

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,
AxEXGRIDLib._IGridEvents_RemoveItemEvent e)
{
}
```

C++

```
void OnRemoveItem(long Item)
{
}
```

C++ Builder

```
void __fastcall RemoveItem(TObject *Sender,Exgridlib_tlb::HITEM Item)
{
}
```

Delphi

```
procedure RemoveItem(ASender: TObject; Item : HITEM);
begin
end;
```

Delphi 8 (.NET only)

```
procedure RemoveItem(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_RemoveItemEvent) Handles RemoveItem
End Sub
```

VB6

```
Private Sub RemoveItem(ByVal Item As EXGRIDLibCtl.HITEM)
End Sub
```

VBA

```
Private Sub RemoveItem(ByVal Item As Long)
End Sub
```

VFP

```
LPARAMETERS Item
```

Xbas…

```
PROCEDURE OnRemoveItem(oGrid,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.Grid.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.EXGRIDLib.ScrollBarEnum
ScrollBar,exontrol.EXGRIDLib.ScrollPartEnum ScrollPart)
{
}
```

VB

```
Private Sub ScrollButtonClick(ByVal sender As System.Object,ByVal ScrollBar As
exontrol.EXGRIDLib.ScrollBarEnum,ByVal ScrollPart As
exontrol.EXGRIDLib.ScrollPartEnum) Handles ScrollButtonClick
End Sub
```

Syntax for ScrollButtonClick event, **/COM** version, on:

C#

```
private void ScrollButtonClick(object sender,
AxEXGRIDLib._IGridEvents_ScrollButtonClickEvent e)
{
}
```

C++

```
void OnScrollButtonClick(long ScrollBar,long ScrollPart)
{
}
```

C++ Builder

```
void __fastcall ScrollButtonClick(TObject *Sender,Exgridlib_tlb::ScrollBarEnum
ScrollBar,Exgridlib_tlb::ScrollPartEnum ScrollPart)
{
}
```

Delphi

```
procedure ScrollButtonClick(ASender: TObject; ScrollBar :
ScrollBarEnum;ScrollPart : ScrollPartEnum);
begin
end;
```

Delphi 8 (.NET only)

```
procedure ScrollButtonClick(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_ScrollButtonClickEvent) Handles ScrollButtonClick
End Sub
```

VB6

```
Private Sub ScrollButtonClick(ByVal ScrollBar As
EXGRIDLibCtl.ScrollBarEnum,ByVal ScrollPart As EXGRIDLibCtl.ScrollPartEnum)
End Sub
```

VBA

```
Private Sub ScrollButtonClick(ByVal ScrollBar As Long,ByVal ScrollPart As Long)
End Sub
```

VFP

```
LPARAMETERS ScrollBar,ScrollPart
```

Xbas…

```
PROCEDURE OnScrollButtonClick(oGrid,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.Grid.1::ScrollBarEnum,ScrollPart as
OLE::Exontrol.Grid.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 Grid1
   .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 Grid1_ScrollButtonClick(ByVal ScrollPart As EXGRIDLibCtl.ScrollPartEnum)
  MsgBox (ScrollPart)
End Sub
```

The following VB.NET sample displays the identifier of the scroll's button being clicked:

```
With AxGrid1
  .BeginUpdate()
  .ScrollBars = EXGRIDLib.ScrollBarsEnum.exDisableBoth
  .set_ScrollPartVisible(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exLeftB1Part Or EXGRIDLib.ScrollPartEnum.exRightB1Part, True)
  .set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exLeftB1Part, "<img></img>1")
  .set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exRightB1Part, "<img></img>2")
  .EndUpdate()
End With
```

```
Private Sub AxGrid1_ScrollButtonClick(ByVal sender As System.Object, ByVal e As
AxEXGRIDLib._IGridEvents_ScrollButtonClickEvent) Handles AxGrid1.ScrollButtonClick
    MessageBox.Show( e.scrollPart.ToString())
End Sub
```

The following C# sample displays the identifier of the scroll's button being clicked:

```
axGrid1.BeginUpdate();
axGrid1.ScrollBars = EXGRIDLib.ScrollBarsEnum.exDisableBoth;
axGrid1.set_ScrollPartVisible(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exLeftB1Part | EXGRIDLib.ScrollPartEnum.exRightB1Part, true);
axGrid1.set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exLeftB1Part , "<img></img>1");
axGrid1.set_ScrollPartCaption(EXGRIDLib.ScrollBarEnum.exVScroll,
EXGRIDLib.ScrollPartEnum.exRightB1Part, "<img></img>2");
axGrid1.EndUpdate();
```

```
private void axGrid1_ScrollButtonClick(object sender,
AxEXGRIDLib._IGridEvents_ScrollButtonClickEvent e)
{
   MessageBox.Show(e.scrollPart.ToString());
}
```

The following C++ sample displays the identifier of the scroll's button being clicked:

```
m_grid.BeginUpdate();
m_grid.SetScrollBars( 15 /*exDisableBoth*/ );
m_grid.SetScrollPartVisible( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ | 32
/*exRightB1Part*/, TRUE );
m_grid.SetScrollPartCaption( 0 /*exVScroll*/, 32768  /*exLeftB1Part*/ , _T("<img>
</img>1") );
m_grid.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("<img></img>2")
);
m_grid.EndUpdate();
```

```
void OnScrollButtonClickGrid1(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.Grid1
   .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). 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. Use the CellValue property to retrieve the cell's value. Use the Selected property to specify whether a column is selected when the FullRowSelect property is exRectSel. The SelectPos property selects/unselects items by position. The Selection property selects/unselects items by index.

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(oGrid)
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 Grid1_SelectionChanged()
    On Error Resume Next
    Dim h As HITEM
    Dim i As Long, j As Long, nCols As Long, nSels As Long
    nCols = Grid1.Columns.Count
    With Grid1.Items
    nSels = .SelectCount
    For i = 0 To nSels - 1
        Dim s As String
        For j = 0 To nCols - 1
            s = s + .CellValue(.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 Grid1_SelectionChanged()
   Grid1.Items.ExpandItem(Grid1.Items.SelectedItem()) = True
End Sub
```

The following VB sample displays the selected items:

```
Private Sub Grid1_SelectionChanged()
   Dim i As Long
   With Grid1.Items
      For i = 0 To .SelectCount - 1
         Debug.Print .CellValue(.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 OnSelectionChangedGrid1()
{
   CItems items = m_grid.GetItems();
   for ( long i = 0; i < items.GetSelectCount(); i++ )
   {
```

```
        COleVariant vtItem( items.GetSelectedItem( i ) );
        CString strOutput;
        strOutput.Format( "%s\n", V2S( &items.GetCellValue( vtItem, COleVariant( (long)0 ) ) )
);
        OutputDebugString( strOutput );
    }
}
```

The following VB.NET sample displays the selected items:

```
Private Sub AxGrid1_SelectionChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles AxGrid1.SelectionChanged
    With AxGrid1.Items
        Dim i As Integer
        For i = 0 To .SelectCount - 1
            Debug.WriteLine(.CellValue(.SelectedItem(i), 0))
        Next
    End With
End Sub
```

The following C# sample displays the selected items:

```
private void axGrid1_SelectionChanged(object sender, System.EventArgs e)
{
    for ( int i = 0; i < axGrid1.Items.SelectCount; i++ )
    {
        object cell = axGrid1.Items.get_CellValue( axGrid1.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.Grid1.Items
    for i = 0 to .SelectCount - 1
        .DefaultItem = .SelectedItem( i )
        wait window nowait .CellValue( 0, 0 )
```

```
    next
endwith
```

event Sort ()

Fired when the control sorts a column.

Type	Description

The control fires the Sort event when the control sorts a column (the user clicks the column's head) or when the sorting position is changed in the control's sort bar. Use the SortOnClick property to specify the action that control executes when the user clicks the column's head. Use the SortBarVisible property to show the control's sort bar. Use the SortOrder property to sorts a column at runtime. Use the SortPosition property to determine the position of the column in the sorting columns collection. Use the ItemBySortPosition property to access a column giving its position in the sorting columns collection. Use the Sort event to sort the data when the SortOnClk property is exUserSort. Use the SingleSort property to allow sorting by single or multiple columns. The EnsureOnSort property prevents scrolling the control's content when the user sorts items.

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(oGrid)
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 Grid1_Sort()
   Dim s As String, i As Long, c As Column
   i = 0
   With Grid1.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 OnSortGrid1()
{
   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 AxGrid1_Sort(ByVal sender As Object, ByVal e As System.EventArgs) Handles
AxGrid1.Sort
   With AxGrid1
      Dim s As String, i As Integer, c As EXGRIDLib.Column
      i = 0
      With AxGrid1.Columns
         c = .ItemBySortPosition(i)
         While (Not c Is Nothing)
            s = s + """" & c.Caption & """ " & IIf(c.SortOrder =
EXGRIDLib.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 axGrid1_Sort(object sender, System.EventArgs e)
{
   string strOutput = "";
   int i = 0;
   EXGRIDLib.Column column = axGrid1.Columns.get_ItemBySortPosition( i );
```

```
    while ( column != null )
    {
        strOutput += column.Caption + " " + ( column.SortOrder ==
EXGRIDLib.SortOrderEnum.SortAscending ? "A" : "D" ) + " ";
        column = axGrid1.Columns.get_ItemBySortPosition( ++i );
    }
    Debug.WriteLine( strOutput );
}
```

event ToolTip (Item as HITEM, ColIndex as Long, Visible as Boolean, X as Long, Y as Long, CX as Long, CY as Long)

Fired when the control prepares the object's tooltip.

Type	Description
Item as HITEM	A long expression that indicates the item's handle or 0 if the cursor is not over the cell.
ColIndex as Long	A long expression that indicates the column's index.
Visible as Boolean	A boolean expression that indicates whether the object's tooltip is visible.
X as Long	A long expression that indicates the left location of the tooltip window. The x values is always expressed in screen coordinates.
Y as Long	A long expression that indicates the top location of the tooltip window. The y values is always expressed in screen coordinates.
CX as Long	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, AxEXGRIDLib._IGridEvents_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,Exgridlib_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:
AxEXGRIDLib._IGridEvents_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
AxEXGRIDLib._IGridEvents_ToolTipEvent) Handles ToolTip
End Sub
```

VB6

```
Private Sub ToolTip(ByVal Item As EXGRIDLibCtl.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(oGrid,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.Grid.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

event URChange (Operation as Long)

Occurs once the control's undo/redo queue is changed.

Type	Description
Operation as Long	A long expression that specifies the type of operation that occurred as defined: • exUndoRedo (0x10, 16), specifies that an undo/redo operation occurred • exUndo(0x11, 17), specifies that an undo operation occurred • exUndo(0x12, 18), specifies that an redo operation occurred

---

The URChange event occurs once an undo/redo operation occurs. The AllowUndoRedo property enables or disables the control's Undo/Redo feature. The UndoListAction property lists the Undo actions that can be performed in the control. The RedoListAction property lists the Redo actions that can be performed in the control. The Undo and Redo features let you remove or repeat single or multiple actions, but all actions must be undone or redone in the order you did or undid them you cant skip actions. For example, if you change the value of three cells in an item and then decide you want to undo the first change you made, you must undo all three changes. To undo an action you need to press Ctrl+Z, while for to redo something you've undone, press Ctrl+Y.

event UserEditorClose (Object as Object, Item as HITEM, ColIndex as Long)

Fired the user editor is about to be opened.

Type	Description
Object as Object	An object created by UserEditor property.
Item as HITEM	A long expression that determines the item's handle. If the Item parameter is 0, and the ColIndex property is different than zero, the ColIndex indicates the handle of the cell where the state is changed.
ColIndex as Long	A long expression that indicates the column's index, if the Item parameter is not zero, a long expression that indicates the handle of the cell if the Item parameter is 0.

Use the UserEditorClose event to notify your application when the user editor is hidden. Use the UserEditorClose event to update the cell's value when user editor is hidden. The control fires UserEditorOleEvent event each time when a an user editor object fires an event.

Syntax for UserEditorClose event, **/NET** version, on:

C#
```
private void UserEditorClose(object sender,object Obj,int Item,int ColIndex)
{
}
```

VB
```
Private Sub UserEditorClose(ByVal sender As System.Object,ByVal Obj As
Object,ByVal Item As Integer,ByVal ColIndex As Integer) Handles UserEditorClose
End Sub
```

Syntax for UserEditorClose event, **/COM** version, on:

C#
```
private void UserEditorClose(object sender,
AxEXGRIDLib._IGridEvents_UserEditorCloseEvent e)
{
}
```

C++
```
void OnUserEditorClose(LPDISPATCH Object,long Item,long ColIndex)
{
}
```

C++ Builder

```
void __fastcall UserEditorClose(TObject *Sender,IDispatch *Object,Exgridlib_tlb::HITEM
Item,long ColIndex)
{
}
```

Delphi

```
procedure UserEditorClose(ASender: TObject; Object : IDispatch;Item :
HITEM;ColIndex : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure UserEditorClose(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_UserEditorCloseEvent);
begin
end;
```

Powe…

```
begin event UserEditorClose(oleobject Object,long Item,long ColIndex)
end event UserEditorClose
```

VB.NET

```
Private Sub UserEditorClose(ByVal sender As System.Object, ByVal e As
AxEXGRIDLib._IGridEvents_UserEditorCloseEvent) Handles UserEditorClose
End Sub
```

VB6

```
Private Sub UserEditorClose(ByVal Object As Object,ByVal Item As
EXGRIDLibCtl.HITEM,ByVal ColIndex As Long)
End Sub
```

VBA

```
Private Sub UserEditorClose(ByVal Object As Object,ByVal Item As Long,ByVal
ColIndex As Long)
End Sub
```

VFP

```
LPARAMETERS Object,Item,ColIndex
```

Xbas…

```
PROCEDURE OnUserEditorClose(oGrid,Object,Item,ColIndex)
RETURN
```

Syntax for UserEditorClose event, **/COM** version (others), on:

Java...
```
<SCRIPT EVENT="UserEditorClose(Object,Item,ColIndex)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc...
```
<SCRIPT LANGUAGE="VBScript">
Function UserEditorClose(Object,Item,ColIndex)
End Function
</SCRIPT>
```

Visual Data...
```
Procedure OnComUserEditorClose Variant llObject HITEM llItem Integer llColIndex
    Forward Send OnComUserEditorClose llObject llItem llColIndex
End_Procedure
```

Visual Objects
```
METHOD OCX_UserEditorClose(Object,Item,ColIndex) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_UserEditorClose(COM _Object,int _Item,int _ColIndex)
{
}
```

XBasic
```
function UserEditorClose as v (Object as P,Item as
OLE::Exontrol.Grid.1::HITEM,ColIndex as N)
end function
```

dBASE
```
function nativeObject_UserEditorClose(Object,Item,ColIndex)
return
```

The following VB sample updates the cell's value when the user editor is hidden (the sample handles the event for an exMaskEdit inside):

```
Private Sub Grid1_UserEditorClose(ByVal Object As Object, ByVal Item As
EXGRIDLibCtl.HITEM, ByVal ColIndex As Long)
   With Grid1.Items
      .CellValue(Item, ColIndex) = Object.Text
   End With
End Sub
```

The following C++ sample updates the cell's value when the user editor is hidden (the sample handles the event for an exMaskEdit inside):

```
#import <maskedit.dll>

#include "Items.h"

void OnUserEditorCloseGrid1(LPDISPATCH Object, long Item, long ColIndex)
{
   MaskEditLib::IMaskEditPtr spMaskEdit( Object );
   if ( spMaskEdit != NULL )
   {
      COleVariant vtNewValue(spMaskEdit->GetText());
      m_grid.GetItems().SetCellValue( COleVariant( Item ), COleVariant( ColIndex ),
vtNewValue );
   }
}
```

where the #import <maskedit.dll> defines the type library of the exMaskEdit component, in the MaskEditLib namespace. The V2S function converts a VARIANT value to a string value and may look like follows:

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
   if ( pv )
   {
      if ( pv->vt == VT_ERROR )
         return szDefault;

      COleVariant vt;
      vt.ChangeType( VT_BSTR, pv );
      return V_BSTR( &vt );
   }
   return szDefault;
}
```

The following VB.NET sample updates the cell's value when the user editor is hidden (the sample handles the event for an exMaskEdit inside):

```
Private Sub AxGrid1_UserEditorClose(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_UserEditorCloseEvent) Handles AxGrid1.UserEditorClose
   With AxGrid1.Items
      .CellValue(e.item, e.colIndex) = e.object.Text
   End With
End Sub
```

The following C# sample updates the cell's value when the user editor is hidden (the sample handles the event for an exMaskEdit inside):

```
private void axGrid1_UserEditorClose(object sender,
AxEXGRIDLib._IGridEvents_UserEditorCloseEvent e)
{
   MaskEditLib.MaskEdit maskEdit = e.@object as MaskEditLib.MaskEdit;
   if (maskEdit != null)
      axGrid1.Items.set_CellValue(e.item, e.colIndex, maskEdit.Text);
}
```

where the MaskEditLib class is defined by adding a new reference to the ExMaskEdit component to your project.

The following VFP sample updates the cell's value when the user editor is hidden (the sample handles the event for an exMaskEdit inside):

```
*** ActiveX Control Event ***
LPARAMETERS object, item, colindex

with thisform.Grid1.Items
   .DefaultItem = item
   .CellValue( 0, colindex ) = object.Text()
endwith
```

event UserEditorOleEvent (Object as Object, Ev as OleEvent, CloseEditor as Boolean, Item as HITEM, ColIndex as Long)

Occurs when an user editor fires an event.

Type	Description
Object as Object	An object created by the UserEditor property.
Ev as OleEvent	An OleEvent object that holds information about the event
CloseEditor as Boolean	A boolean expression that indicates whether the control should close the user editor.
Item as HITEM	A long expression that determines the item's handle. If the Item parameter is 0, and the ColIndex property is different than zero, the ColIndex indicates the handle of the cell where the state is changed.
ColIndex as Long	A long expression that indicates the column's index, if the Item parameter is not zero, a long expression that indicates the handle of the cell if the Item parameter is 0.

The UserEditorOleEvent is fired every time when an user editor object fires an event. The information about fired event is stored in the Ev parameter. The CloseEditor parameter is useful to inform the control when the editor should be hidden, on certain events. The control fires the UserEditorOpen event when a ActiveX editor is about to be shown. The control fires the UserEditorClose event when the user editor is hidden.

Syntax for UserEditorOleEvent event, **/NET** version, on:

C#

```
private void UserEditorOleEvent(object sender,object
Obj,exontrol.EXGRIDLib.OleEvent Ev,ref bool CloseEditor,int Item,int ColIndex)
{
}
```

VB

```
Private Sub UserEditorOleEvent(ByVal sender As System.Object,ByVal Obj As
Object,ByVal Ev As exontrol.EXGRIDLib.OleEvent,ByRef CloseEditor As
Boolean,ByVal Item As Integer,ByVal ColIndex As Integer) Handles
UserEditorOleEvent
End Sub
```

Syntax for UserEditorOleEvent event, **/COM** version, on:

C#

```
private void UserEditorOleEvent(object sender,
```

```
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent e)
{
}
```

C++

```
void OnUserEditorOleEvent(LPDISPATCH Object,LPDISPATCH Ev,BOOL FAR*
CloseEditor,long Item,long ColIndex)
{
}
```

C++ Builder

```
void __fastcall UserEditorOleEvent(TObject *Sender,IDispatch
*Object,Exgridlib_tlb::IOleEvent *Ev,VARIANT_BOOL *
CloseEditor,Exgridlib_tlb::HITEM Item,long ColIndex)
{
}
```

Delphi

```
procedure UserEditorOleEvent(ASender: TObject; Object : IDispatch;Ev :
IOleEvent;var CloseEditor : WordBool;Item : HITEM;ColIndex : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure UserEditorOleEvent(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent);
begin
end;
```

Powe…

```
begin event UserEditorOleEvent(oleobject Object,oleobject Ev,boolean
CloseEditor,long Item,long ColIndex)
end event UserEditorOleEvent
```

VB.NET

```
Private Sub UserEditorOleEvent(ByVal sender As System.Object, ByVal e As
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent) Handles UserEditorOleEvent
End Sub
```

VB6

```
Private Sub UserEditorOleEvent(ByVal Object As Object,ByVal Ev As
EXGRIDLibCtl.IOleEvent,CloseEditor As Boolean,ByVal Item As
EXGRIDLibCtl.HITEM,ByVal ColIndex As Long)
End Sub
```

VBA

```
Private Sub UserEditorOleEvent(ByVal Object As Object,ByVal Ev As Object,CloseEditor As Boolean,ByVal Item As Long,ByVal ColIndex As Long)
End Sub
```

VFP

```
LPARAMETERS Object,Ev,CloseEditor,Item,ColIndex
```

Xbas…

```
PROCEDURE OnUserEditorOleEvent(oGrid,Object,Ev,CloseEditor,Item,ColIndex)
RETURN
```

Syntax for UserEditorOleEvent event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="UserEditorOleEvent(Object,Ev,CloseEditor,Item,ColIndex)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function UserEditorOleEvent(Object,Ev,CloseEditor,Item,ColIndex)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComUserEditorOleEvent Variant llObject Variant llEv Boolean llCloseEditor HITEM llItem Integer llColIndex
    Forward Send OnComUserEditorOleEvent llObject llEv llCloseEditor llItem llColIndex
End_Procedure
```

Visual Objects

```
METHOD OCX_UserEditorOleEvent(Object,Ev,CloseEditor,Item,ColIndex) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_UserEditorOleEvent(COM _Object,COM _Ev,COMVariant /*bool*/ _CloseEditor,int _Item,int _ColIndex)
{
}
```

XBasic

```
function UserEditorOleEvent as v (Object as P,Ev as
OLE::Exontrol.Grid.1::IOleEvent,CloseEditor as L,Item as
OLE::Exontrol.Grid.1::HITEM,ColIndex as N)
end function
```

dBASE

```
function nativeObject_UserEditorOleEvent(Object,Ev,CloseEditor,Item,ColIndex)
return
```

The following VB sample closes the editor and focus a new column when the user presses the TAB key:

```
Private Sub Grid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Item As EXGRIDLibCtl.HITEM, ByVal
ColIndex As Long)
   If (Ev.Name = "KeyDown") Then
      Dim iKey As Long
      iKey = Ev(0).Value
      If iKey = vbKeyTab Then
         With Grid1
            CloseEditor = True
            .FocusColumnIndex = .FocusColumnIndex + 1
            .SearchColumnIndex = .FocusColumnIndex
         End With
      End If
   End If
End Sub
```

The following VB sample closes the Exontrol.ComboBox user editor when the user selects a new value, or when it presses the Escape key. Also the sample changes the value of the cell in the control:

```
Private Sub Grid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Item As EXGRIDLibCtl.HITEM, ByVal
ColIndex As Long)
   ' Closes the Exontrol.ComboBox when user changes the value in the control
   If nEvents = 0 Then
```

```
    If (Ev.Name = "Change") Then
        With Grid1
            .BeginUpdate
            With .Items
                .CellValue(Item, ColIndex) = Object.Select(1)
                .CellValueFormat(Item, ColIndex) = exHTML
                .CellValue(Item, ColIndex) = .CellValue(Item, ColIndex) + " <fgcolor=FF0000>
[<b>changed</b>]</fgcolor>"
            End With
            .EndUpdate
        End With
        CloseEditor = True
    End If

    If (Ev.Name = "KeyPress") Then
        Dim l As Long
        l = Ev(0).Value
        If l = vbKeyEscape Then
            CloseEditor = True
        End If
    End If
  End If
End Sub
```

The following VB sample displays the event and its parameters when an user editor object fires an event:

```
Private Sub Grid1_UserEditorOleEvent(ByVal Object As Object, ByVal Ev As
EXGRIDLibCtl.IOleEvent, CloseEditor As Boolean, ByVal Item As EXGRIDLibCtl.HITEM, ByVal
ColIndex As Long)
  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 C++ sample displays the event and its parameters when an user editor object fires an event:

```
#import <exgrid.dll> rename( "GetItems", "exGetItems" )

void OnUserEditorOleEventGrid1(LPDISPATCH Object, LPDISPATCH Ev, BOOL FAR*
CloseEditor, long Item, long ColIndex)
{
  EXGRIDLib::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++ )
    {
      EXGRIDLib::IOleEventParamPtr spParam = spEvent->GetParam( COleVariant( i ) );
      strOutput.Format( "Name: %s, Value: %s\n", spParam->Name.operator const char *
(), V2S( &spParam->Value ) );
      OutputDebugString( strOutput );
    }
  }
  OutputDebugString( "" );
}
```

where the #import<grid.dll> defines the EXGRIDLib namespace that exports definitions for the OleEvent and OleEventParam objects. The V2S function converts a VARIANT value to a string value and may look like follows:

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
  if ( pv )
```

```
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}
```

The following VB.NET sample displays the event and its parameters when an user editor object fires an event:

```
Private Sub AxGrid1_UserEditorOleEvent(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent) Handles AxGrid1.UserEditorOleEvent
    Debug.WriteLine("Event's name: " & e.ev.Name)
    Dim i As Long
    For i = 0 To e.ev.CountParam - 1
        Dim eP As EXGRIDLib.OleEventParam
        eP = e.ev(i)
        Debug.WriteLine("Name: " & e.ev.Name & " Value: " & eP.Value)
    Next
End Sub
```

The following C# sample displays the event and its parameters when an user editor object fires an event:

```
private void axGrid1_UserEditorOleEvent(object sender,
AxEXGRIDLib._IGridEvents_UserEditorOleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine("Event's name: " + e.ev.Name.ToString());
    for (int i = 0; i < e.ev.CountParam; i++)
    {
        EXGRIDLib.IOleEventParam evP = e.ev[i];
        System.Diagnostics.Debug.WriteLine("Name: " + evP.Name.ToString() + ", Value: " +
evP.Value.ToString());
    }
```

```
}
```

The following VFP sample displays the event and its parameters when an user editor object fires an event:

```
*** ActiveX Control Event ***
LPARAMETERS object, ev, closeeditor, item, colindex

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 UserEditorOpen (Object as Object, Item as HITEM, ColIndex as Long)

Occurs when an user editor is about to be opened.

Type	Description
Object as Object	An object created by UserEditor property
Item as HITEM	A long expression that determines the item's handle. If the Item parameter is 0, and the ColIndex property is different than zero, the ColIndex indicates the handle of the cell where the state is changed.
ColIndex as Long	A long expression that indicates the column's index, if the Item parameter is not zero, a long expression that indicates the handle of the cell if the Item parameter is 0.

The control supports custom ActiveX editors support. The control fires the UserEditorOpen event when an user editor is shown. Use the UserEditorOpen event to initialize the user editor when it is shown. For instance, if you have a custom maskedit control you can initialize the mask and the value based on the cell's value property. Use the CellValue property to access the cell's value. The control fires the UserEditorOleEvent event each time when an user editor fires an event. The control fires the UserEditorClose event when an user editor is hidden.

Syntax for UserEditorOpen event, **/NET** version, on:

C#
```
private void UserEditorOpen(object sender,object Obj,int Item,int ColIndex)
{
}
```

VB
```
Private Sub UserEditorOpen(ByVal sender As System.Object,ByVal Obj As
Object,ByVal Item As Integer,ByVal ColIndex As Integer) Handles UserEditorOpen
End Sub
```

Syntax for UserEditorOpen event, **/COM** version, on:

C#
```
private void UserEditorOpen(object sender,
AxEXGRIDLib._IGridEvents_UserEditorOpenEvent e)
{
}
```

C++

```
void OnUserEditorOpen(LPDISPATCH Object,long Item,long ColIndex)
{
}
```

C++ Builder

```
void __fastcall UserEditorOpen(TObject *Sender,IDispatch
*Object,Exgridlib_tlb::HITEM Item,long ColIndex)
{
}
```

Delphi

```
procedure UserEditorOpen(ASender: TObject; Object : IDispatch;Item :
HITEM;ColIndex : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure UserEditorOpen(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_UserEditorOpenEvent);
begin
end;
```

Powe…

```
begin event UserEditorOpen(oleobject Object,long Item,long ColIndex)
end event UserEditorOpen
```

VB.NET

```
Private Sub UserEditorOpen(ByVal sender As System.Object, ByVal e As
AxEXGRIDLib._IGridEvents_UserEditorOpenEvent) Handles UserEditorOpen
End Sub
```

VB6

```
Private Sub UserEditorOpen(ByVal Object As Object,ByVal Item As
EXGRIDLibCtl.HITEM,ByVal ColIndex As Long)
End Sub
```

VBA

```
Private Sub UserEditorOpen(ByVal Object As Object,ByVal Item As Long,ByVal
ColIndex As Long)
End Sub
```

VFP

```
LPARAMETERS Object,Item,ColIndex
```

Xbas…

```
PROCEDURE OnUserEditorOpen(oGrid,Object,Item,ColIndex)
RETURN
```

Syntax for UserEditorOpen event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="UserEditorOpen(Object,Item,ColIndex)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function UserEditorOpen(Object,Item,ColIndex)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComUserEditorOpen Variant llObject HITEM llItem Integer llColIndex
    Forward Send OnComUserEditorOpen llObject llItem llColIndex
End_Procedure
```

Visual Objects

```
METHOD OCX_UserEditorOpen(Object,Item,ColIndex) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_UserEditorOpen(COM _Object,int _Item,int _ColIndex)
{
}
```

XBasic

```
function UserEditorOpen as v (Object as P,Item as
OLE::Exontrol.Grid.1::HITEM,ColIndex as N)
end function
```

dBASE

```
function nativeObject_UserEditorOpen(Object,Item,ColIndex)
return
```

The following VB sample selects an item into an user editor of EXCOMBOBOXLibCtl.ComboBox type (the sample uses the Exontrol's ExComboBox Component):

```
Private Sub Grid1_UserEditorOpen(ByVal Object As Object, ByVal Item As
EXGRIDLibCtl.HITEM, ByVal ColIndex As Long)
  On Error Resume Next
```

```
    nEvents = nEvents + 1
    'Selects the value in the combo box
    With Object          ' Points to an EXCOMBOBOXLibCtl.ComboBox object
        Dim sID As String
        sID = Grid1.Items.CellValue(Item, ColIndex)
        If (Grid1.Items.CellValueFormat(Item, ColIndex) = exHTML) Then
            sID = Mid(sID, 1, InStr(1, sID, " ", vbTextCompare) - 1)
        End If
        .Select(1) = sID
        If .Items.SelectCount > 0 Then
            .Items.EnsureVisibleItem .Items.SelectedItem(0)
        End If
    End With
    nEvents = nEvents - 1
End Sub
```

The following samples use the Exontrol's ExMaskEdit Component to mask floating point numbers using digit grouping.

The following VB sample initializes the mask's value when user editor is shown (the sample calls the Text property of the exMaskEdit component):

```
Private Sub Grid1_UserEditorOpen(ByVal Object As Object, ByVal Item As
EXGRIDLibCtl.HITEM, ByVal ColIndex As Long)
    With Grid1.Items
        Object.Text = .CellValue(Item, ColIndex)
    End With
End Sub
```

The following C++ sample initializes the mask's value when user editor is shown:

```
#import <maskedit.dll>

#include "Items.h"

void OnUserEditorOpenGrid1(LPDISPATCH Object, long Item, long ColIndex)
{
    MaskEditLib::IMaskEditPtr spMaskEdit( Object );
    if ( spMaskEdit != NULL )
```

```
    {
        CString strValue = V2S( &m_grid.GetItems().GetCellValue( COleVariant( Item ),
COleVariant( ColIndex ) ) );
        spMaskEdit->PutText( strValue.AllocSysString() );
    }
}
```

where the #import <maskedit.dll> defines the type library of the exMaskEdit component, in the MaskEditLib namespace. The V2S function converts a VARIANT value to a string value and may look like follows:

```
static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}
```

The following VB.NET sample initializes the mask's value when user editor is shown:

```
Private Sub AxGrid1_UserEditorOpen(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_UserEditorOpenEvent) Handles AxGrid1.UserEditorOpen
    With AxGrid1.Items
        e.object.Text = .CellValue(e.item, e.colIndex)
    End With
End Sub
```

The following C# sample initializes the mask's value when user editor is shown:

```
private void axGrid1_UserEditorOpen(object sender,
AxEXGRIDLib._IGridEvents_UserEditorOpenEvent e)
{
```

```
    MaskEditLib.MaskEdit maskEdit = e.@object as MaskEditLib.MaskEdit;
    if (maskEdit != null)
    {
        object cellValue = axGrid1.Items.get_CellValue(e.item, e.colIndex);
        maskEdit.Text = (cellValue != null ? cellValue.ToString() : "");
    }
}
```

where the MaskEditLib class is defined by adding a new reference to the ExMaskEdit component to your project.

The following VFP sample initializes the mask's value when user editor is shown:

```
*** ActiveX Control Event ***
LPARAMETERS object, item, colindex

with thisform.Grid1.Items
    .DefaultItem = item
    object.Text = .CellValue( 0, colindex )
endwith
```

event ValidateValue (Item as HITEM, ColIndex as Long, NewValue as Variant, Cancel as Boolean)

Occurs before user changes the cell's value.

Type	Description
Item as HITEM	A long expression that determines the item's handle. If the Item parameter is 0, and the ColIndex property is different than zero, the ColIndex indicates the handle of the cell where the change occurs.
ColIndex as Long	A long expression that indicates the column's index, if the Item parameter is not zero, a long expression that indicates the handle of the cell if the Item parameter is 0.
NewValue as Variant	A Variant value that indicates the value being validated.
Cancel as Boolean	A boolean expression that indicates whether the value is valid or not. By default, the Cancel parameter is False, and so the NewValue parameter is valid. If the Cancel parameter is set on True, the control considers the NewValue being a non valid value, so the Change event is not fired.

The ValidateValue event notifies your application that the user is about to change the cell's value using the control's UI. Use the ValidateValue event to prevent users enter wrong values to the cells. The ValidateValue event is fired **only** if the CauseValidateValue property is not zero and the user alters the focused value. The validation can be done per cell or per item, in other words, the validation can be made if the user leaves the focused cell, or focused item. If the Cancel parameter is True, the user can't move the focus to a new cell/item, until the Cancel parameter is False. If the Cancel parameter is False the control fires the Change event to notify your application that the cell's value is changed. Use the Edit method to programmatically edit the focused cell. Call the DiscardValidateValue method to restore back the values being changed during the validation.

During ValidateValue event, the Items.CellValue(Item,ColIndex) and Items.CellCaption(Item,ColIndex) properties retrieve the original value/caption of the cell. You can access the modified value for any cell in the validating item using the Items.CellValue(-1,ColIndex) and Items.CellCaption(-1,ColIndex), or uses the -1 identifier for the Item parameter of the Items.CellValue and Items.CellCaption properties.

During the validation you may have the following order of the events:

- Edit - prevent showing the editor for specified cell.
- EditOpen - indicates that the editor for the focused cell is being opened.

- EditClose - indicates that the editor for the focused cell is being closed.
- ValidateValue - notifies your application that the value must be validated (Cancel parameter on False)
- Change - notifies the application once the user validates the newly value. In case the control is bounded to a database, the change is performed to the database too.
- Error - notifies the application for any error (for instance, if the change is not supported by the database, the Error indicates the error being issued).

The ValidateValue event is not fired if the CellValue property is called during the event.

Syntax for ValidateValue event, **/NET** version, on:

C#
```
private void ValidateValue(object sender,int Item,int ColIndex,object NewValue,ref bool Cancel)
{
}
```

VB
```
Private Sub ValidateValue(ByVal sender As System.Object,ByVal Item As Integer,ByVal ColIndex As Integer,ByVal NewValue As Object,ByRef Cancel As Boolean) Handles ValidateValue
End Sub
```

Syntax for ValidateValue event, **/COM** version, on:

C#
```
private void ValidateValue(object sender, AxEXGRIDLib._IGridEvents_ValidateValueEvent e)
{
}
```

C++
```
void OnValidateValue(long Item,long ColIndex,VARIANT NewValue,BOOL FAR* Cancel)
{
}
```

C++ Builder
```
void __fastcall ValidateValue(TObject *Sender,Exgridlib_tlb::HITEM Item,long ColIndex,Variant NewValue,VARIANT_BOOL * Cancel)
{
}
```

Delphi
```
procedure ValidateValue(ASender: TObject; Item : HITEM;ColIndex :
```

```
Integer;NewValue : OleVariant;var Cancel : WordBool);
begin
end;
```

Delphi 8 (.NET only)

```
procedure ValidateValue(sender: System.Object; e:
AxEXGRIDLib._IGridEvents_ValidateValueEvent);
begin
end;
```

Powe…

```
begin event ValidateValue(long Item,long ColIndex,any NewValue,boolean Cancel)
end event ValidateValue
```

VB.NET

```
Private Sub ValidateValue(ByVal sender As System.Object, ByVal e As
AxEXGRIDLib._IGridEvents_ValidateValueEvent) Handles ValidateValue
End Sub
```

VB6

```
Private Sub ValidateValue(ByVal Item As EXGRIDLibCtl.HITEM,ByVal ColIndex As
Long,ByVal NewValue As Variant,Cancel As Boolean)
End Sub
```

VBA

```
Private Sub ValidateValue(ByVal Item As Long,ByVal ColIndex As Long,ByVal
NewValue As Variant,Cancel As Boolean)
End Sub
```

VFP

```
LPARAMETERS Item,ColIndex,NewValue,Cancel
```

Xbas…

```
PROCEDURE OnValidateValue(oGrid,Item,ColIndex,NewValue,Cancel)
RETURN
```

Syntax for ValidateValue event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="ValidateValue(Item,ColIndex,NewValue,Cancel)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function ValidateValue(Item,ColIndex,NewValue,Cancel)
```

```
End Function
</SCRIPT>
```

Visual Data...

```
Procedure OnComValidateValue HITEM llItem Integer llColIndex Variant
llNewValue Boolean llCancel
   Forward Send OnComValidateValue llItem llColIndex llNewValue llCancel
End_Procedure
```

Visual Objects

```
METHOD OCX_ValidateValue(Item,ColIndex,NewValue,Cancel) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_ValidateValue(int _Item,int _ColIndex,COMVariant
_NewValue,COMVariant /*bool*/ _Cancel)
{
}
```

XBasic

```
function ValidateValue as v (Item as OLE::Exontrol.Grid.1::HITEM,ColIndex as
N,NewValue as A,Cancel as L)
end function
```

dBASE

```
function nativeObject_ValidateValue(Item,ColIndex,NewValue,Cancel)
return
```

The following VB sample asks the user to validate the value for **each** cell that's edited:

```
Private Sub Grid_ValidateValue(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As
Long, ByVal NewValue As Variant, Cancel As Boolean)
   Cancel = True ' Causes all cells in the item to be invalid
   If MsgBox("The ValidateValue event just occurs. Do the change with this value '" &
NewValue & "'?", vbYesNo) = vbYes Then
      Cancel = False ' Only cells where user selects the Yes button, are valid
   End If
   Grid.Edit ' Continue editing a cell.
End Sub
```

The following C++ sample asks the user to validate the value for **each** cell that's edited:

```
CString V2S( const VARIANT* pvtValue )
```

```
{
   COleVariant vt;
   vt.ChangeType( VT_BSTR, (LPVARIANT)pvtValue );
   return V_BSTR( &vt );
}

void OnValidateValueGrid1(long Item, long ColIndex, const VARIANT FAR& NewValue,
BOOL FAR* Cancel)
{
   *Cancel = TRUE;// Causes all cells to be invalid
   if ( MessageBox( "The ValidateValue event just occurs. Do the change with this value '" +
V2S( &NewValue ) + "'?", "Information", MB_YESNO ) == IDYES )
      *Cancel = FALSE; // Only cells where user selects the Yes button, are valid
   COleVariant vtOptional; V_VT(&vtOptional) = VT_ERROR;
   m_grid.Edit( vtOptional ); // Continue editing a cell.
}
```

The following C++ sample asks the user to enter a value greater than 10 on the first column, if the value is less than 10:

```
Private Sub Grid_ValidateValue(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As
Long, ByVal NewValue As Variant, Cancel As Boolean)
   If (ColIndex = 0) Then
      If (NewValue < 10) Then
         MsgBox "Enter a value greater than 10."
         Cancel = True ' Cancels only the cells with the value less than 10.
         Grid.Edit ' Continue editing a cell.
      End If
   End If
End Sub
```

The following VB.NET sample asks the user to validate the values on the first column:

```
Private Sub AxGrid1_ValidateValue(ByVal sender As Object, ByVal e As
AxEXGRIDLib._IGridEvents_ValidateValueEvent) Handles AxGrid1.ValidateValue
   If (e.colIndex = 0) Then
      e.cancel = True
      Dim strMessage As String = "The ValidateValue event just occurs. Do the change with
```

```
this value '" & e.newValue.ToString() & "'?"
        If (MessageBox.Show(strMessage, "Question", MessageBoxButtons.YesNoCancel,
MessageBoxIcon.Question) = Windows.Forms.DialogResult.Yes) Then
            e.cancel = False
        End If
    End If
End Sub
```

The following C# sample asks the user to validate the values on the first column:

```
private void axGrid1_ValidateValue(object sender,
AxEXGRIDLib._IGridEvents_ValidateValueEvent e)
{
    if (e.colIndex == 0)
    {
        e.cancel = true;
        string strMessage = "The ValidateValue event just occurs. Do the change with this
value '" + e.newValue.ToString() + "'?";
        if ( MessageBox.Show(strMessage, "Question", MessageBoxButtons.YesNoCancel,
MessageBoxIcon.Question) == DialogResult.Yes )
            e.cancel = false;
    }
}
```

The following VFP sample asks the user to validate the values on the first column:

```
*** ActiveX Control Event ***
LPARAMETERS item, colindex, newvalue, cancel

with thisform.Grid1.Items
    if ( colindex = 0 )
        cancel = .t.
        local strMessage
        strMessage = "The ValidateValue event just occurs. Do the change with this value '" +
newvalue + "'?"
        if ( MessageBox( strMessage, 3 ) = 6 )
            cancel = .f.
        endif
```

```
    endif
endwith
```

OwnerDrawHandler object

The OwnerDrawHandler interface provides an elegant way to let user paints the cell. The CellOwnerDraw property requires an object that implements the OwnerDrawHandler interface. Use the Def(exCellOwneDraw) property to assign an owner draw object for the entire column. The control calls DrawCell method when an owner draw cell requires painting. The inteface definition is like follows:

```
[
 uuid(BA219E1D-D1CD-4682-81AA-7E1D9D37B187),
 pointer_default(unique)
]
interface IOwnerDrawHandler : IUnknown
{
 [id(1), helpstring("The source paints the cell.")] HRESULT DrawCell( long hDC, long left,
long top, long right, long bottom, long Item, long Column, IDispatch* Source );
 [id(2), helpstring("The source erases the cell's background.")] HRESULT DrawCellBk( long
hDC, VARIANT* Options, long left, long top, long right, long bottom, long Item, long
Column, IDispatch* Source );
}
```

Use the DrawCellBk method to erase the cell's background. The DrawCell method is called before painting the cell's caption.

The following sample shows how to paint a gradient color into the cells:

```
Option Explicit
Implements IOwnerDrawHandler

Private Type RECT
   left As Long
   top As Long
   right As Long
   bottom As Long
End Type
Private Const ETO_OPAQUE = 2
Private Declare Sub InflateRect Lib "user32" (lpRect As RECT, ByVal x As Long, ByVal Y As
Long)
Private Declare Function ExtTextOut Lib "gdi32" Alias "ExtTextOutA" (ByVal hdc As Long,
```

```
ByVal x As Long, ByVal Y As Long, ByVal wOptions As Long, lpRect As RECT, ByVal lpString
As String, ByVal nCount As Long, lpDx As Long) As Long
Private Declare Function SetBkColor Lib "gdi32" (ByVal hdc As Long, ByVal crColor As
Long) As Long
Private Declare Function DrawText Lib "user32" Alias "DrawTextA" (ByVal hdc As Long,
ByVal lpStr As String, ByVal nCount As Long, lpRect As RECT, ByVal wFormat As Long) As
Long
Private Declare Function SetTextColor Lib "gdi32" (ByVal hdc As Long, ByVal crColor As
Long) As Long
Private Declare Function OleTranslateColor Lib "olepro32" (ByVal c As Long, ByVal p As
Long, c As Long) As Long
Private Const DT_VCENTER = &H4
Private Const DT_CENTER = &H1
Private Const DT_WORDWRAP = &H10

Private Sub DrawGradient(ByVal hdc As Long, ByVal left As Long, ByVal top As Long, ByVal
right As Long, ByVal bottom As Long, ByVal c1 As Long, ByVal c2 As Long)
  On Error Resume Next
  Dim x As Long, rg, gg, bg, r1, r2, g1, g2, b1, b2
  Dim rc As RECT
  With rc
    .left = left
    .right = right
    .top = top
    .bottom = bottom
  End With
  OleTranslateColor c1, 0, c1
  OleTranslateColor c2, 0, c2
  r1 = c1 Mod 256
  r2 = c2 Mod 256
  b1 = Int(c1 / 65536)
  b2 = Int(c2 / 65536)
  g1 = Int(c1 / 256) Mod 256
  g2 = Int(c2 / 256) Mod 256
  For x = left To right Step 2
    rc.left = x
    SetBkColor hdc, RGB(r1 + (x - left) * (r2 - r1) / (right - left), g1 + (x - left) * (g2 - g1) /
```

```
(right - left), b1 + (x - left) * (b2 - b1) / (right - left))
        ExtTextOut hdc, rc.left, rc.top, ETO_OPAQUE, rc, " ", 1, x
    Next
    End Sub

Private Sub Form_Load()
    With Grid1
        .BeginUpdate
        .LinesAtRoot = False
        .SortOnClick = False
        .MarkTooltipCells = True
        .ShowFocusRect = False
        .MarkSearchColumn = False
        .ShowFocusRect = True
        .ColumnAutoResize = True
        .BackColor = vbWhite
        .SelBackColor = vbWhite
        .SelForeColor = vbBlue
        Set .Picture = LoadPicture(App.Path + "\exontrol.gif")
        .PictureDisplay = LowerRight
        .SelBackMode = exTransparent
        .SelBackColor = vbWhite

        ' Adds few columns
        With .Columns
            .Add("Name").Width = 242
            With .Add("Description")
                .Width = 356
                .HeaderImage = 2
                .Editor.EditType = MemoType
                .Editor.Appearance = RaisedApp
            End With
        End With

        ' Adds few items
        With .Items
            Dim h As HITEM, h2 As HITEM, h3 As HITEM
```

```
        h = .AddItem("My Desktop")
        .CellBold(h, 0) = True
        ' Defines the cell that becomes the title for the divider
        .ItemHeight(h) = .ItemHeight(h) + 4
        .ItemDivider(h) = 0
        .CellBackColor(h) = &HFF6531
        .ItemForeColor(h) = vbWhite
        .ItemDividerLine(h) = EmptyLine
        Set .CellOwnerDraw(h, 0) = Me

        h2 = .InsertItem(h, , "Hard Disk Drives")
        .CellBold(h2, 0) = True
        .ItemDivider(h2) = 0
        .ItemDividerLine(h2) = DotLine
        .CellBackColor(h2) = vbBlue
        .ItemHeight(h2) = .ItemHeight(h2) + 4
        .CellForeColor(h2, 0) = &HFF6531
        .CellForeColor(h2, 0) = vbWhite
        Set .CellOwnerDraw(h2, 0) = Me

        h3 = .InsertItem(h2, , "Scratch (C:)" & vbCrLf & "1.95 GB" & vbCrLf)
        .CellPicture(h3, 0) = LoadPicture(App.Path + "\hard.gif")
        .CellSingleLine(h3, 0) = False
        .CellValue(h3, 1) = "You can add hardware devices to your Windows CEbased target
platform that are not directly supported by Windows CE. However, if you do, you must
supply device drivers for the additional devices."
        .CellSingleLine(h3, 1) = False
        .CellToolTip(h3, 0) = "This is a bit of text that shoud appear when the cursor is over
a cell."

        h3 = .InsertItem(h2, , "Main (E:)" & vbCrLf & "15 GB" & vbCrLf)
        .CellPicture(h3, 0) = LoadPicture(App.Path + "\hard.gif")
        .CellForeColor(h3, 0) = RGB(128, 128, 128)
        .CellSingleLine(h3, 0) = False
        .CellValue(h3, 1) = "Windows CE versions 1.01 and later provide kernel support to
enable stream interface drivers to access additional built-in hardware devices."
        .CellSingleLine(h3, 1) = False
```

```
.CellBackColor(h3, 1) = RGB(196, 196, 196)
.CellForeColor(h3, 1) = vbBlack
Set .CellOwnerDraw(h3, 1) = Me

.ExpandItem(h2) = True

h2 = .InsertItem(h, , "Devices with Removable Storage")
.CellBold(h2, 0) = True
.ItemDivider(h2) = 0
.ItemDividerLine(h2) = DotLine
.CellBackColor(h2) = vbBlue
.ItemHeight(h2) = .ItemHeight(h2) + 4
.CellForeColor(h2, 0) = vbWhite
  Set .CellOwnerDraw(h2, 0) = Me

h3 = .InsertItem(h2, , vbCrLf & "3″ Floppy (A:)" & vbCrLf)
.CellPicture(h3, 0) = LoadPicture(App.Path + "\floppy.gif")
.CellSingleLine(h3, 0) = False
With .CellEditor(h3, 1)
   .EditType = ColorType
End With
.CellValue(h3, 1) = .CellBackColor(.ItemParent(h3), 0)
.CellData(h3, 1) = True

h3 = .InsertItem(h2, , vbCrLf & "CD Reader" & vbCrLf)
.CellPicture(h3, 0) = LoadPicture(App.Path + "\floppy.gif")
.CellSingleLine(h3, 0) = False
With .CellEditor(h3, 1)
   .EditType = ColorType
End With
.CellValue(h3, 1) = .CellBackColor(.ItemParent(h3), 0)
.CellData(h3, 1) = True

.ExpandItem(h2) = True

.ExpandItem(h) = True
```

```
    h = .AddItem("Folder Options")
    .CellBold(h, 0) = True
    .ItemDivider(h) = 0
    .CellBackColor(h) = &HFF6531
    .ItemForeColor(h) = vbWhite
    .ItemHeight(h) = .ItemHeight(h) + 4
    Set .CellOwnerDraw(h, 0) = Me

    h2 = .InsertItem(h, , "Web View")
    .CellImage(h2, 0) = 2
    .CellBold(h2, 0) = True
    .ItemDivider(h2) = 0
    .ItemDividerLine(h2) = DotLine
    .ItemHeight(h2) = .ItemHeight(h2) + 4
    .CellForeColor(h2, 0) = vbWhite
    .CellBackColor(h2) = vbBlue
    Set .CellOwnerDraw(h2, 0) = Me

    h3 = .InsertItem(h2, , "Enable Web content in folders")
    .CellHasRadioButton(h3, 0) = True
    .CellImage(h3, 0) = 1
    .CellRadioGroup(h3, 0) = 1234
    .CellState(h3, 0) = 1
    .CellEditorVisible(h3, 1) = False

    h3 = .InsertItem(h2, , "Use Windows Classic folders")
    .CellHasRadioButton(h3, 0) = True
    .CellRadioGroup(h3, 0) = 1234
    .CellImage(h3, 0) = 2
    .CellEditorVisible(h3, 1) = False

    .ExpandItem(h2) = True

    .ExpandItem(h) = True

End With
.EndUpdate
```

```
    End With
End Sub

Private Sub Grid1_Change(ByVal Item As EXGRIDLibCtl.HITEM, ByVal ColIndex As Long,
NewValue As Variant)
    With Grid1.Items
        If .CellData(Item, ColIndex) Then
            .CellBackColor(.ItemParent(Item), 0) = NewValue
        End If
    End With
End Sub

Private Sub IOwnerDrawHandler_DrawCellBk(ByVal hDC As Long, Options As Variant, ByVal
left As Long, ByVal top As Long, ByVal right As Long, ByVal bottom As Long, ByVal Item As
Long, ByVal Column As Long, ByVal Source As Object)
End Sub

Private Sub IOwnerDrawHandler_DrawCell(ByVal hdc As Long, ByVal left As Long, ByVal
top As Long, ByVal right As Long, ByVal bottom As Long, ByVal Item As Long, ByVal
Column As Long, ByVal Source As Object)
    With Source.Items
        ' Draws the background cell by gradient
        DrawGradient hdc, left, top, right / 2, bottom, vbWhite, .CellBackColor(Item, Column)
        DrawGradient hdc, right / 2, top, right, bottom, .CellBackColor(Item, Column),
vbWhite

        ' Gets the caption cell
        Dim str As String
        str = .CellValue(Item, Column)

        ' Draws the caption cell
        Dim rc As RECT
        With rc
            .left = left
            .right = right
            .top = top
            .bottom = bottom
```

```
        End With

        SetTextColor hdc, .CellForeColor(Item, Column)
        rc.top = rc.top + 2
        DrawText hdc, str, Len(str), rc, DT_CENTER Or DT_WORDWRAP
    End With
End Sub
```

The following sample erase the cell's background, but let the control paints the cell's content:

```
Implements IOwnerDrawHandler

Private Declare Function MoveToEx Lib "gdi32" (ByVal hDC As Long, ByVal x As Long, ByVal Y As Long, lpPoint As POINTAPI) As Long
Private Declare Function LineTo Lib "gdi32" (ByVal hDC As Long, ByVal x As Long, ByVal Y As Long) As Long
Private Type RECT
    left As Long
    top As Long
    right As Long
    bottom As Long
End Type
Private Const ETO_OPAQUE = 2
Private Declare Sub InflateRect Lib "user32" (lpRect As RECT, ByVal x As Long, ByVal Y As Long)
Private Declare Function ExtTextOut Lib "gdi32" Alias "ExtTextOutA" (ByVal hDC As Long, ByVal x As Long, ByVal Y As Long, ByVal wOptions As Long, lpRect As RECT, ByVal lpString As String, ByVal nCount As Long, lpDx As Long) As Long
Private Declare Function SetBkColor Lib "gdi32" (ByVal hDC As Long, ByVal crColor As Long) As Long
Private Declare Function DrawText Lib "user32" Alias "DrawTextA" (ByVal hDC As Long, ByVal lpStr As String, ByVal nCount As Long, lpRect As RECT, ByVal wFormat As Long) As Long
Private Declare Function SetTextColor Lib "gdi32" (ByVal hDC As Long, ByVal crColor As Long) As Long
Private Declare Function OleTranslateColor Lib "olepro32" (ByVal c As Long, ByVal p As Long, c As Long) As Long
```

```
Private Const DT_VCENTER = &H4
Private Const DT_CENTER = &H1
Private Const DT_WORDWRAP = &H10
Private Const DT_SINGLELINE = &H20

Private Type POINTAPI
    x As Long
    Y As Long
End Type

Private Sub DrawGradient(ByVal hDC As Long, ByVal left As Long, ByVal top As Long, ByVal
right As Long, ByVal bottom As Long, ByVal c1 As Long, ByVal c2 As Long)
  On Error Resume Next
  Dim x As Long, rg, gg, bg, r1, r2, g1, g2, b1, b2
  Dim rc As RECT
  With rc
    .left = left
    .right = right
    .top = top
    .bottom = bottom
  End With
  OleTranslateColor c1, 0, c1
  OleTranslateColor c2, 0, c2
  r1 = c1 Mod 256
  r2 = c2 Mod 256
  b1 = Int(c1 / 65536)
  b2 = Int(c2 / 65536)
  g1 = Int(c1 / 256) Mod 256
  g2 = Int(c2 / 256) Mod 256
  For x = left To right Step 2
    rc.left = x
    SetBkColor hDC, RGB(r1 + (x - left) * (r2 - r1) / (right - left), g1 + (x - left) * (g2 - g1) /
(right - left), b1 + (x - left) * (b2 - b1) / (right - left))
    ExtTextOut hDC, rc.left, rc.top, ETO_OPAQUE, rc, " ", 1, x
  Next
End Sub
```

```
Private Sub Form_Load()
   With Grid1.Items
      Set .CellOwnerDraw(.FindItem("Root 2"), 0) = Me
   End With
End Sub

Private Sub IOwnerDrawHandler_DrawCell(ByVal hDC As Long, ByVal left As Long, ByVal
top As Long, ByVal right As Long, ByVal bottom As Long, ByVal Item As Long, ByVal
Column As Long, ByVal Source As Object)
 End Sub

Private Sub IOwnerDrawHandler_DrawCellBk(ByVal hDC As Long, Options As Variant, ByVal
left As Long, ByVal top As Long, ByVal right As Long, ByVal bottom As Long, ByVal Item As
Long, ByVal Column As Long, ByVal Source As Object)
   Dim c1 As Long, c2 As Long, c As Long
   c1 = Source.BackColor
   c2 = Source.SelBackColor
   DrawGradient hDC, left, top, (right + left) / 2, bottom, c1, c2
   DrawGradient hDC, (right + left) / 2, top, right, bottom, c2, c1
End Sub
```

Name	Description

Expressions

An expression is a string which defines a formula or criteria, that's evaluated at runtime. The expression may be a combination of variables, constants, strings, dates and operators/functions. For instance 1000 format `` gets 1,000.00 for US format, while 1.000,00 is displayed for German format.

The Exontrol's **eXPression** component is a syntax-editor that helps you to define, view, edit and evaluate expressions. Using the eXPression component you can easily view or check if the expression you have used is syntactically correct, and you can evaluate what is the result you get giving different values to be tested. The Exontrol's eXPression component can be used as an user-editor, to configure your applications.

Usage examples:

- 100 + 200, adds numbers and returns 300
- "100" + 200, concatenates the strings, and returns "100200"
- currency(1000) displays the value in currency format based on the current regional setting, such as "$1,000.00" for US format.
- 1000 format `` gets 1,000.00 for English format, while 1.000,00 is displayed for German format
- 1000 format `2|.|3|,` always gets 1,000.00 no matter of settings in the control panel.
- upper("string") converts the giving string in uppercase letters, such as "STRING"
- date(dateS('3/1/' + year(9:=#1/1/2018#)) + ((1:=(((255 - 11 * (year(=:9) mod 19)) - 21) mod 30) + 21) + (=:1 > 48 ? -1 : 0) + 6 - ((year(=:9) + int(year(=:9) / 4)) + =:1 + (=:1 > 48 ? -1 : 0) + 1) mod 7)) returns the date the Easter Sunday will fall, for year 2018. In this case the expression returns #4/1/2018#. If #1/1/2018# is replaced with #1/1/2019#, the expression returns #4/21/2019#.

Listed bellow are all predefined constants, operators and functions the general-expression supports:

The constants can be represented as:

- numbers in **decimal** format (where dot character specifies the decimal separator). For instance: -1, 100, 20.45, .99 and so on
- numbers in **hexa-decimal** format (preceded by **0x** or **0X** sequence), uses sixteen distinct symbols, most often the symbols 0-9 to represent values zero to nine, and A, B, C, D, E, F (or alternatively a, b, c, d, e, f) to represent values ten to fifteen. Hexadecimal numerals are widely used by computer system designers and programmers. As each hexadecimal digit represents four binary digits (bits), it allows a more human-friendly representation of binary-coded values. For instance, 0xFF,

0x00FF00, and so so.

- **date-time** in format **#mm/dd/yyyy hh:mm:ss#**, For instance, #1/31/2001 10:00# means the January 31th, 2001, 10:00 AM
- **string**, if it starts / ends with any of the **'** or ` or **"** characters. If you require the starting character inside the string, it should be escaped ( preceded by a \ character ). For instance, `Mihai`, "Filimon", 'has', "\"a quote\"", and so on

The predefined constants are:

- **bias** (BIAS constant), defines the difference, in minutes, between Coordinated Universal Time (UTC) and local time. For example, Middle European Time (MET, GMT+01:00) has a time zone bias of "-60" because it is one hour ahead of UTC. Pacific Standard Time (PST, GMT-08:00) has a time zone bias of "+480" because it is eight hours behind UTC. For instance, date(value - bias/24/60) converts the UTC time to local time, or date(date('now') + bias/24/60) converts the current local time to UTC time. For instance, "date(value - bias/24/60)" converts the value date-time from UTC to local time, while "date(value + bias/24/60)" converts the local-time to UTC time.
- **dpi** (DPI constant), specifies the current DPI setting. and it indicates the minimum value between **dpix** and **dpiy** constants. For instance, if current DPI setting is 100%, the dpi constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpi returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%
- **dpix** (DPIX constant), specifies the current DPI setting on x-scale. For instance, if current DPI setting is 100%, the dpix constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpix returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%
- **dpiy** (DPIY constant), specifies the current DPI setting on x-scale. For instance, if current DPI setting is 100%, the dpiy constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpiy returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%

The supported binary arithmetic operators are:

- ***** (multiplicity operator), priority 5
- **/** (divide operator), priority 5
- **mod** (reminder operator), priority 5
- **+** (addition operator), priority 4 (concatenates two strings, if one of the operands is of string type)
- **-** (subtraction operator), priority 4

The supported unary boolean operators are:

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

The supported binary boolean operators are:

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

The supported binary boolean operators, all these with the same priority 0, are :

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

The supported binary range operators, all these with the same priority 5, are :

- a **MIN** b (min operator), indicates the minimum value, so a **MIN** b returns the value of a, if it is less than b, else it returns b. For instance, the expression value MIN 10 returns always a value greater than 10.
- a **MAX** b (max operator), indicates the maximum value, so a **MAX** b returns the value of a, if it is greater than b, else it returns b. For instance, the expression value MAX 100 returns always a value less than 100.

The supported binary operators, all these with the same priority 0, are :

- **:= (Store operator),** stores the result of expression to variable. The syntax for := operator is

  variable := expression

  where variable is a integer between 0 and 9. You can use the **=:** operator to restore any stored variable (please make the difference between := and =:). For instance, *(0:=dbl(value)) = 0 ? "zero" : =:0*, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the **:=** and **=:** are two distinct operators, the first for storing the result into a variable, while the second for restoring the variable

- **=: (Restore operator),** restores the giving variable (previously saved using the store operator). The syntax for =: operator is

  =: variable

  where variable is a integer between 0 and 9. You can use the **:=** operator to store the value of any expression (please make the difference between := and =:). For

instance, *(0:=dbl(value)) = 0 ? "zero" : =:0*, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the **:=** and **=:** are two distinct operators, the first for storing the result into a variable, while the second for restoring the variable

The supported ternary operators, all these with the same priority 0, are :

- **?** (**Immediate If operator**), returns and executes one of two expressions, depending on the evaluation of an expression. The syntax for *?* operator is

  expression ? true_part : false_part

  , while it executes and returns the true_part if the expression is true, else it executes and returns the false_part. For instance, the *%0 = 1 ? 'One' : (%0 = 2 ? 'Two' : 'not found')* returns 'One' if the value is 1, 'Two' if the value is 2, and 'not found' for any other value. A n-ary equivalent operation is the case() statement, which is available in newer versions of the component.

The supported n-ary operators are (with priority 5):

- ***array*** *(at operator),* returns the element from an array giving its index (0 base). The *array* operator returns empty if the element is found, else the associated element in the collection if it is found. The syntax for *array* operator is

  expression array (c1,c2,c3,...cn)

  , where the c1, c2, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the *month(value)-1 array ('J','F','M','A','M','Jun','J','A','S','O','N','D')* is equivalent with *month(value)-1 case (default:''; 0:'J';1:'F';2:'M';3:'A';4:'M';5:'Jun';6:'J';7:'A';8:'S';9:'O';10:'N';11:'D')*.

- ***in*** *(include operator),* specifies whether an element is found in a set of constant elements. The *in* operator returns -1 (True) if the element is found, else 0 (false) is retrieved. The syntax for *in* operator is

  expression in (c1,c2,c3,...cn)

  , where the c1, c2, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the *value in (11,22,33,44,13)* is equivalent with *(expression = 11) or (expression = 22) or (expression = 33) or (expression = 44) or (expression = 13)*. The *in* operator is not a time consuming as the equivalent *or* version is, so when you have large number of constant elements it is recommended using the *in* operator. Shortly, if the collection of elements has 1000 elements the *in* operator could take up to 8 operations in order to find if an element fits the set, else if the *or*

statement is used, it could take up to 1000 operations to check, so by far, the *in* operator could save time on finding elements within a collection.

- ***switch*** *(switch operator),* returns the value being found in the collection, or a predefined value if the element is not found (default). The syntax for *switch* operator is

expression switch (default,c1,c2,c3,...,cn)

, where the c1, c2, ... are constant elements, and the default is a constant element being returned when the element is not found in the collection. The constant elements could be numeric, date or string expressions. The equivalent syntax is "%0 = c 1 ? c 1 : (%0 = c 2 ? c 2 : (... ? . : default))". The *switch* operator is very similar with the *in* operator excepts that the first element in the switch is always returned by the statement if the element is not found, while the returned value is the value itself instead -1. For instance, the *%0 switch ('not found',1,4,7,9,11)* gets 1, 4, 7, 9 or 11, or 'not found' for any other value. As the *in* operator the *switch* operator uses binary searches for fitting the element, so it is quicker that iif (immediate if operator) alterative.

- ***case()*** *(case operator)* returns and executes one of n expressions, depending on the evaluation of the expression (IIF - immediate IF operator is a binary case() operator). The syntax for *case()* operator is:

expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)

If the default part is missing, the case() operator returns the value of the expression if it is not found in the collection of cases (c1, c2, ...). For instance, if the value of expression is not any of c1, c2, ... the default_expression is executed and returned. If the value of the expression is c1, then the *case()* operator executes and returns the *expression1.* The *default, c1, c2, c3, ...* must be constant elements as numbers, dates or strings. For instance, the *date(shortdate(value)) case (default:0 ; #1/1/2002#:1 ; #2/1/2002#:1; #4/1/2002#:1; #5/1/2002#:1)* indicates that only *#1/1/2002#, #2/1/2002#, #4/1/2002# and #5/1/2002#* dates returns 1, since the others returns 0. For instance the following sample specifies the hour being non-working for specified dates: *date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)* statement indicates the working hours for dates as follows:

  - #4/1/2009#, from hours 06:00 AM to 12:00 PM
  - #4/5/2009#, from hours 07:00 AM to 10:00 AM and hours 03:00PM, 04:00PM, 06:00PM and 10:00PM
  - #5/1/2009#, from hours 12:00 AM to 08:00 AM

The *in*, *switch* and *case()* use binary search to look for elements so they are faster then using iif and or expressions. Obviously, the priority of the operations inside the expression is determined by ( ) parenthesis and the priority for each operator.

The supported conversion unary operators are:

- **type** (unary operator) retrieves the type of the object. The type operator may return any of the following: 0 - empty (not initialized), 1 - null, 2 - short, 3 - long, 4 - float, 5 - double, 6 - currency, **7 - date**, **8 - string**, 9 - object, 10 - error, **11 - boolean**, 12 - variant, 13 - any, 14 - decimal, 16 - char, 17 - byte, 18 - unsigned short, 19 - unsigned long, 20 - long on 64 bits, 21 - unsigned long on 64 bites. For instance *type(%1) = 8* specifies the cells (on the column with the index 1) that contains string values.
- **str** (unary operator) converts the expression to a string. The str operator converts the expression to a string. For instance, the *str(-12.54)* returns the string "-12.54".
- **dbl** (unary operator) converts the expression to a number. The dbl operator converts the expression to a number. For instance, the *dbl("12.54")* returns 12.54
- **date** (unary operator) converts the expression to a date, based on your regional settings. For instance, the *date(``)* gets the current date (no time included), the *date(`now`)* gets the current date-time, while the *date("01/01/2001")* returns #1/1/2001#
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS. For instance, the *dateS("01/01/2001 14:00:00")* returns #1/1/2001 14:00:00#
- **hex** (unary operator) converts the giving string from hexa-representation to a numeric value, or converts the giving numeric value to hexa-representation as string. For instance, hex(`FF`) returns 255, while the hex(255) or hex(0xFF) returns the `FF` string. The hex(hex(`FFFFFFFF`)) always returns `FFFFFFFF` string, as the second hex call converts the giving string to a number, and the first hex call converts the returned number to string representation (hexa-representation).

The bitwise operators for numbers are:

- a **bitand** b (binary operator) computes the AND operation on bits of a and b, and returns the unsigned value. For instance, 0x01001000 bitand 0x10111000 returns 0x00001000.
- a **bitor** b (binary operator) computes the OR operation on bits of a and b, and returns the unsigned value. For instance, 0x01001000 bitor 0x10111000 returns 0x11111000.
- a **bitxor** b (binary operator) computes the XOR (exclusive-OR) operation on bits of a and b, and returns the unsigned value. For instance, 0x01110010 bitxor 0x10101010 returns 0x11011000.
- a **bitshift** (b) (binary operator) shifts every bit of a value to the left if b is negative, or to the right if b is positive, for b times, and returns the unsigned value. For instance, 128 bitshift 1 returns 64 (dividing by 2) or 128 bitshift (-1) returns 256 (multiplying by

2)

- **bitnot** (unary operator) flips every bit of x, and returns the unsigned value. For instance, bitnot(0x00FF0000) returns 0xFF00FFFF.

The operators for numbers are:

- **int** (unary operator) retrieves the integer part of the number. For instance, the *int(12.54)* returns 12
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2. For instance, the *round(12.54)* returns 13
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument. For instance, the *floor(12.54)* returns 12
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2. For instance, the *abs(-12.54)* returns 12.54
- **sin** (unary operator) returns the sine of an angle of x radians. For instance, the *sin(3.14)* returns 0.001593.
- **cos** (unary operator) returns the cosine of an angle of x radians. For instance, the *cos(3.14)* returns -0.999999.
- **asin** (unary operator) returns the principal value of the arc sine of x, expressed in radians. For instance, the *2*asin(1)* returns the value of PI.
- **acos** (unary operator) returns the principal value of the arc cosine of x, expressed in radians. For instance, the *2*acos(0)* returns the value of PI
- **sqrt** (unary operator) returns the square root of x. For instance, the *sqrt(81)* returns 9.
- **currency** (unary operator) formats the giving number as a currency string, as indicated by the control panel. For instance, *currency(value)* displays the value using the current format for the currency ie, 1000 gets displayed as $1,000.00, for US format.
- value **format** 'flags' (binary operator) formats the value with specified flags. If flags is empty, the number is displayed as shown in the field "Number" in the "Regional and Language Options" from the Control Panel. For instance the *1000 format ''* displays 1,000.00 for English format, while 1.000,00 is displayed for German format. 1000 format '2|.|3|,' will always displays 1,000.00 no matter of settings in the control panel. If formatting the number fails for some invalid parameter, the value is displayed with no formatting.

  The ' flags' for format operator is a list of values separated by | character such as '*NumDigits*|*DecimalSep*|*Grouping*|*ThousandSep*|*NegativeOrder*|*LeadingZero*' with the following meanings:

  - *NumDigits* - specifies the number of fractional digits, If the flag is missing, the field "No. of digits after decimal" from "Regional and Language Options" is using.
  - *DecimalSep* - specifies the decimal separator. If the flag is missing, the field "Decimal symbol" from "Regional and Language Options" is using.
  - Grouping - indicates the number of digits in each group of numbers to the left of

the decimal separator. Values in the range 0 through 9 and 32 are valid. The most significant grouping digit indicates the number of digits in the least significant group immediately to the left of the decimal separator. Each subsequent grouping digit indicates the next significant group of digits to the left of the previous group. If the last value supplied is not 0, the remaining groups repeat the last group. Typical examples of settings for this member are: 0 to group digits as in 123456789.00; 3 to group digits as in 123,456,789.00; and 32 to group digits as in 12,34,56,789.00. If the flag is missing, the field "Digit grouping" from "Regional and Language Options" indicates the grouping flag.

- *ThousandSep* - specifies the thousand separator. If the flag is missing, the field "Digit grouping symbol" from "Regional and Language Options" is using.
- *NegativeOrder* - indicates the negative number mode. If the flag is missing, the field "Negative number format" from "Regional and Language Options" is using. The valid values are 0, 1, 2, 3 and 4 with the following meanings:
  - 0 - Left parenthesis, number, right parenthesis; for example, (1.1)
  - 1 - Negative sign, number; for example, -1.1
  - 2 - Negative sign, space, number; for example, - 1.1
  - 3 - Number, negative sign; for example, 1.1-
  - 4 - Number, space, negative sign; for example, 1.1 -
- *LeadingZero* - indicates if leading zeros should be used in decimal fields. If the flag is missing, the field "Display leading zeros" from "Regional and Language Options" is using. The valid values are 0, 1

The operators for strings are:

- **len** (unary operator) retrieves the number of characters in the string. For instance, the *len("Mihai")* returns 5.
- **lower** (unary operator) returns a string expression in lowercase letters. For instance, the *lower("MIHAI")* returns "mihai"
- **upper** (unary operator) returns a string expression in uppercase letters. For instance, the *upper("mihai")* returns "MIHAI"
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names. For instance, the *proper("mihai")* returns "Mihai"
- **ltrim** (unary operator) removes spaces on the left side of a string. For instance, the *ltrim(" mihai")* returns "mihai"
- **rtrim** (unary operator) removes spaces on the right side of a string. For instance, the *rtrim("mihai ")* returns "mihai"
- **trim** (unary operator) removes spaces on both sides of a string. For instance, the *trim(" mihai ")* returns "mihai"
- **reverse** (unary operator) reverses the order of the characters in the string a. For instance, the *reverse("Mihai")* returns "iahiM"
- a **startwith** b (binary operator) specifies whether a string starts with specified string (

0 if not found, -1 if found). For instance *"Mihai" startwith "Mi"* returns -1

- a **endwith** b (binary operator) specifies whether a string ends with specified string (0 if not found, -1 if found). For instance *"Mihai" endwith "ai"* returns -1
- a **contains** b (binary operator) specifies whether a string contains another specified string (0 if not found, -1 if found). For instance *"Mihai" contains "ha"* returns -1
- a **left** b (binary operator) retrieves the left part of the string. For instance *"Mihai" left 2* returns "Mi".
- a **right** b (binary operator) retrieves the right part of the string. For instance *"Mihai" right 2* returns "ai"
- a **lfind** b (binary operator) The a lfind b (binary operator) searches the first occurrence of the string b within string a, and returns -1 if not found, or the position of the result (zero-index). For instance *"ABCABC" lfind "C"* returns 2
- a **rfind** b (binary operator) The a rfind b (binary operator) searches the last occurrence of the string b within string a, and returns -1 if not found, or the position of the result (zero-index). For instance *"ABCABC" rfind "C"* returns 5.
- a **mid** b (binary operator) retrieves the middle part of the string a starting from b (1 means first position, and so on). For instance *"Mihai" mid 2* returns "ihai"
- a **count** b (binary operator) retrieves the number of occurrences of the b in a. For instance *"Mihai" count "i"* returns 2.
- a **replace** b **with** c (double binary operator) replaces in a the b with c, and gets the result. For instance, the *"Mihai" replace "i" with ""* returns "Mha" string, as it replaces all "i" with nothing.
- a **split** b (binary operator) splits the a using the separator b, and returns an array. For instance, the *weekday(value) array 'Sun Mon Thu Wed Thu Fri Sat' split ' '* gets the weekday as string. This operator can be used with the array.
- a **like** b (binary operator) compares the string a against the pattern b. The pattern b may contain wild-characters such as *, ?, # or [] and can have multiple patterns separated by space character. In order to have the space, or any other wild-character inside the pattern, it has to be escaped, or in other words it should be preceded by a \ character. For instance value like `F*e` matches all strings that start with F and ends on e, or value like `a* b*` indicates any strings that start with a or b character.
- a **lpad** b (binary operator) pads the value of a to the left with b padding pattern. For instance, 12 lpad "0000" generates the string "0012".
- a **rpad** b (binary operator) pads the value of a to the right with b padding pattern. For instance, 12 lpad "____" generates the string "12__".
- a **concat** b (binary operator) concatenates the a (as string) for b times. For instance, "x" concat 5, generates the string "xxxxx".

The operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. For instance, the *time(#1/1/2001 13:00#)* returns "1:00:00 PM"

- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance, the *timeF(#1/1/2001 13:00#)* returns "13:00:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel. For instance, the *shortdate(#1/1/2001 13:00#)* returns "1/1/2001"
- **shortdateF** (unary operator) formats a date as a date string using the "MM/DD/YYYY" format. For instance, the *shortdateF(#1/1/2001 13:00#)* returns "01/01/2001"
- **dateF** (unary operator) converts the date expression to a string expression in "MM/DD/YYYY HH:MM:SS" format. For instance, the *dateF(#01/01/2001 14:00:00#)* returns #01/01/2001 14:00:00#
- **longdate** (unary operator) formats a date as a date string using the long date format, as specified in the control's panel. For instance, the *longdate(#1/1/2001 13:00#)* returns "Monday, January 01, 2001"
- **year** (unary operator) retrieves the year of the date (100,...,9999). For instance, the *year(#12/31/1971 13:14:15#)* returns 1971
- **month** (unary operator) retrieves the month of the date (1, 2,...,12). For instance, the *month(#12/31/1971 13:14:15#)* returns 12.
- **day** (unary operator) retrieves the day of the date (1, 2,...,31). For instance, the *day(#12/31/1971 13:14:15#)* returns 31
- **yearday** (unary operator) retrieves the number of the day in the year, or the days since January 1st (0, 1,...,365). For instance, the *yearday(#12/31/1971 13:14:15#)* returns 365
- **weekday** (unary operator) retrieves the number of days since Sunday (0 - Sunday, 1 - Monday,..., 6 - Saturday). For instance, the *weekday(#12/31/1971 13:14:15#)* returns 5.
- **hour** (unary operator) retrieves the hour of the date (0, 1, ..., 23). For instance, the *hour(#12/31/1971 13:14:15#)* returns 13
- **min** (unary operator) retrieves the minute of the date (0, 1, ..., 59). For instance, the *min(#12/31/1971 13:14:15#)* returns 14
- **sec** (unary operator) retrieves the second of the date (0, 1, ..., 59). For instance, the *sec(#12/31/1971 13:14:15#)* returns 15

The expression supports also **immediate if** (similar with iif in visual basic, or ? : in C++) ie cond ? value_true : value_false, which means that once that cond is true the value_true is used, else the value_false is used. Also, it supports variables, up to 10 from 0 to 9. For instance, 0:="Abc" means that in the variable 0 is "Abc", and =:0 means retrieves the value of the variable 0. For instance, the len(%0) ? (0:=(%1+%2) ? currency(=:0) else `` ) : `` gets the sum between second and third column in currency format if it is not zero, and only if the first column is not empty. As you can see you can use the variables to avoid computing several times the same thing (in this case the sum %1 and %2 .