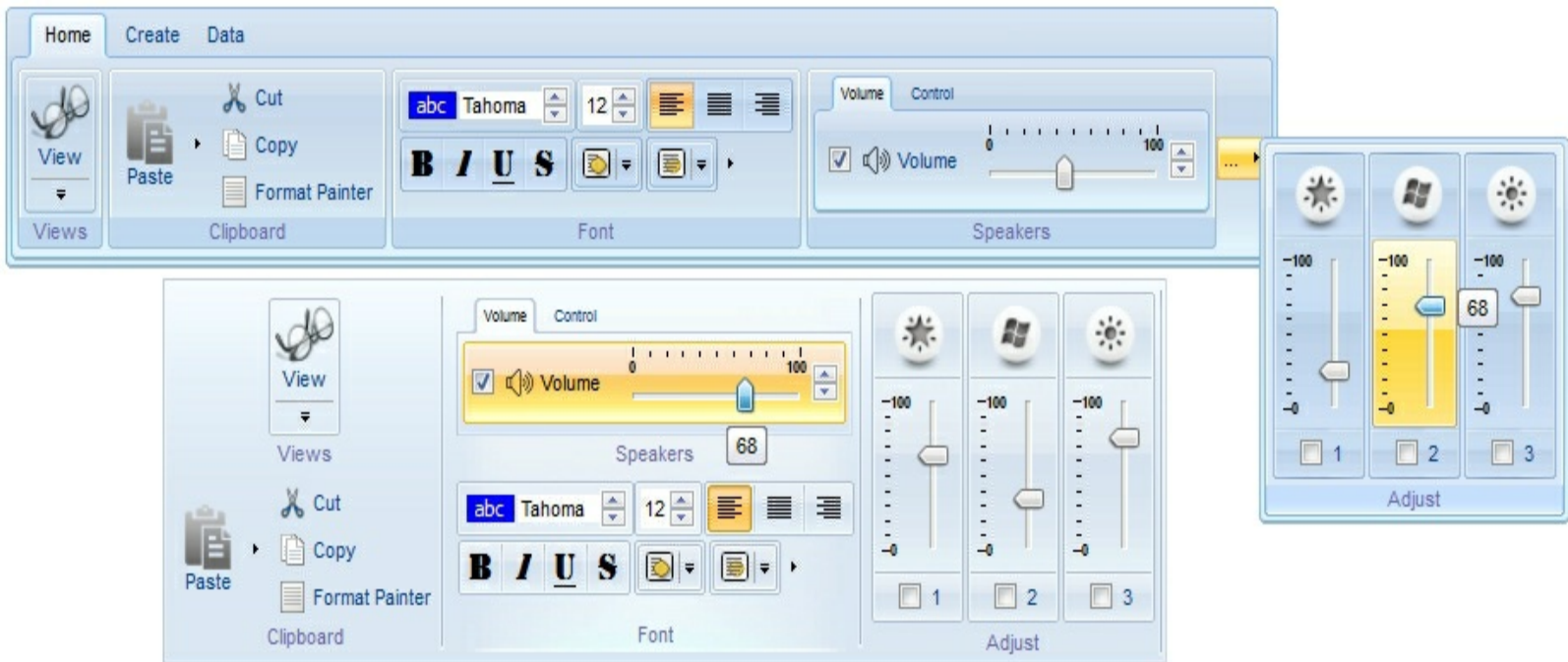


ExRibbon

The eXRibbon component, similar with the Microsoft Office's "Fluent User Interface", or Ribbon, allows you to display graphical control elements in the form of a set of toolbars placed on several tabs. The eXRibbon component is written from scratch, uses [EBN](#) technology to let the user changes its visual appearance using skins, and requires no dependencies to Microsoft Office, Microsoft Ribbon API, or any other third party library. The major difference between our ribbon implementation and others, is that you can display and use built-editors (like sliders, spin, progress, ...) anywhere on the control.

Features include:

- Skinnable Interface support
- Ability to layout the items horizontally or/and vertically
- Single/Multiple Lines HTML support
- Pictures / Images / Icons support
- TAB support, for ribbon items as well
- Check box / Radio button support
- Ability to assign BUTTON, DROP-DOWN BUTTON, EDIT, MASK, COLOR, FONT, SPIN, SLIDER, SCROLLBAR, PROGRESS, ... fields to any item
- Ability to display items as sub-menu or grouped inside the item
- Multi-lines HTML Tooltip support for any item
- Incremental Search/Filter the items on a submenu as you type
- Unlimited options to show any HTML text, images, colors, EBNs, patterns, frames anywhere on the items' background
- Partially Translucent support
- Fade, light-up/down animation effects
- Ability to use any ActiveX control inside sub menus
- Keyboard and Mouse Wheel support
- and more...



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

How to get support?

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

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

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

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

Regards,
Exontrol Development Team

<https://www.exontrol.com>

constants AlignmentEnum


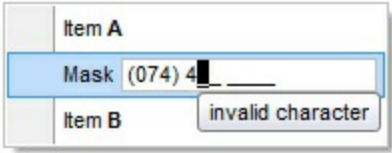
Specifies the object's Alignment. The [Alignment](#) property specifies the item's alignment. The [Caption](#) property supports built-in HTML format, so you can use the <c> to centers the item's caption or <r> to align to the right the item's caption. The AlignmentEnum expression supports the following values:

Name	Value	Description
exLeft	0	Left
exCenter	1	Center
exRight	2	Right

constants AllowEditEnum

The AllowEditEnum type specifies the type of editors that can be associated with the item. The [AllowEdit](#) property associates an editor to the current item. The [EditCaption](#) property specifies the value to show in the edit field. The [EditWidth](#) property specifies the size/width of the edit field inside the item. The [EditBorder](#) property specifies whether the edit shows a border around it. The [EditOption](#) property specifies different options to be used for a specified edit field. The control fires the [EditChange](#) event when the user changes the edit's caption. Use the [ShowLocalPopup](#) property to provide a drop down list. The [ShowAsButton](#) property specifies the whether the current item displays a button or a select button (drop down).

Currently, the supported editors are:

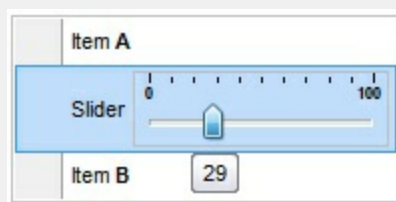
Name	Value	Description
exlItemDisableEdit	0	No editor is assigned to the current item.
exlItemEditText	1	<div><p>A text-box editor is assigned to the current item. The exlItemEditText can be combined with the exlItemEditReadOnly or exlItemEditSpin flags.</p></div>
exlItemEditMask	2	<div><p>A masked text-box editor is assigned to the current item. The EditMask property specifies the mask of the edit field. The EditValue property specifies the value of the edit field, without the masking characters. The EditOption(exEditMaskFloat) specifies whether the edit field mask a floating/decimal/integer point number. The exlItemEditMask can be combined with the exlItemEditReadOnly or exlItemEditSpin flags.</p></div>
		<div><p>A slider editor is assigned to the current item. The exlItemEditSlider can be combined with the exlItemEditReadOnly, exlItemEditVertical or exlItemEditSpin flags. The EditValue property</p></div>

indicates the current slider position/value.

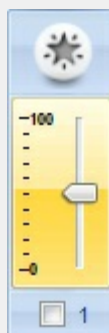
- The [EditOption\(exEditMinValue\)](#) / [EditOption\(exEditMaxValue\)](#) specifies the limits values of the slider editor.
- The [EditOption\(exEditTickStyle\)](#) property specifies the way the ticks are shown on the slider.
- The [EditOption\(exEditTickFrequency\)](#) property specifies the frequency to show the ticks on the slider control.
- The [EditOption\(exEditTickLabel\)](#) property specifies labels to be shown on the slider's ticks.
- The [EditOption\(exEditSmallChange\)](#) property specifies the amount by which the edit's position changes when the user presses an arrow key.
- The [EditOption\(exEditLargeChange\)](#) property specifies the amount by which the edit's position changes when the user presses an CTRL + arrow key.
- The [EditOption\(exEditChangeToolTip\)](#) property specifies the expression that determines the HTML tooltip to be shown when the item's value is changed.

exItemEditSlider

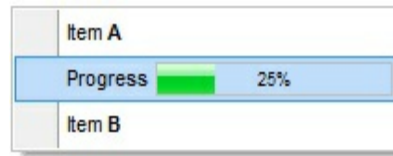
3



If exItemEditSlider flag is combined with the exItemEditVertical you can get:



A progress editor is assigned to the current item. The `exItemEditProgress` can be combined with the `exItemEditReadOnly`, `exItemEditVertical` or `exItemEditSpin` flags. The [EditValue](#) property indicates the current progress position/value.



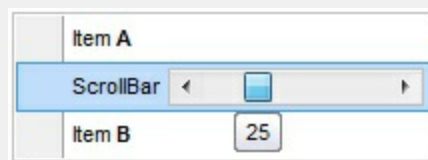
`exItemEditProgress`

4

If `exItemEditProgress` flag is combined with the `exItemEditVertical` you can get:



A scrollbar editor is assigned to the current item. The `exItemEditScrollBar` can be combined with the `exItemEditReadOnly`, `exItemEditVertical` or `exItemEditSpin` flags. The [EditValue](#) property indicates the current scroll position/value.



If `exItemEditScrollBar` flag is combined with the `exItemEditVertical` you can get:



`exItemEditScrollBar`

5

- The [EditOption\(exEditMinValue\)](#) /

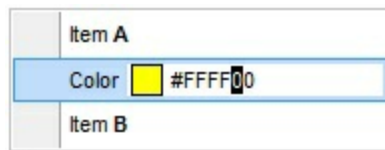
[EditOption\(exEditMaxValue\)](#) specifies the limits values of the scroll editor.

- The [EditOption\(exEditSmallChange\)](#) property specifies the amount by which the edit's position changes when the user presses an arrow key.
- The [EditOption\(exEditLargeChange\)](#) property specifies the amount by which the edit's position changes when the user presses an CTRL + arrow key.
- The [EditOption\(exEditChangeToolTip\)](#) property specifies the expression that determines the HTML tooltip to be shown when the item's value is changed.

exItemEditColor

6

A color editor is assigned to the current item. The exItemEditColor can be combined with the exItemEditReadOnly or exItemEditSpin flags. The [EditValue](#) property indicates the current color value.



exItemEditFont

7

A font editor is assigned to the current item. The exItemEditFont can be combined with the exItemEditReadOnly or exItemEditSpin flags. The [EditCaption](#) property indicates the current font name.



exItemEditReadOnly

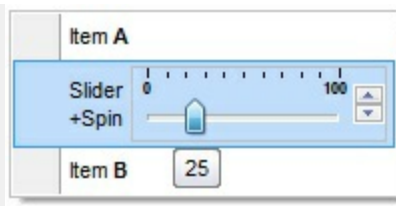
256

Disables the current's item editor. This flag can be combined with any other option.

A spin editor is assigned to the current item. This flag can be combined with any other option. The following picture combines a exItemEditSlider with the exItemEditSpin

exItemEditSpin

512



- The [EditOption\(exEditSpinStep\)](#) specifies the step to advance when user clicks the editor's spin.

exItemEditVertical

1024

The editor is vertically oriented. You can combine this flag with exItemEditSlider, exItemEditProgress and exItemEditScrollBar

constants AppearanceEnum

The AppearanceEnum type specifies the visual appearance/border of the control or popup menus. The [Appearance](#) property specifies the visual appearance/border of the control. The [PopupAppearance](#) property specifies the default visual appearance/border of the popup menus. The [LocalAppearance](#) property specifies the default visual appearance/border of the drop down menu to be shown when selecting a drop down button. The AppearanceEnum type supports the following values:

Name	Value	Description
NoBorder	0	NoBorder
FlatBorder	1	FlatBorder
SunkenBorder	2	SunkenBorder
RaisedBorder	3	RaisedBorder
EtchedBorder	4	EtchedBorder
BumpBorder	5	BumpBorder
ShadowBorder	6	ShadowBorder
InsetBorder	7	InsetBorder
SingleBorder	8	SingleBorder

constants BackgroundPartEnum

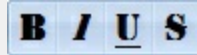
The BackgroundPartEnum type indicates parts in the control. Use the [Background](#) property to specify a background color or a visual appearance for specific parts in the control. A Color expression that indicates the background color for a specified part. The last 7 bits in the high significant byte 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.

Name	Value	Description
exToolTipAppearance	64	Specifies the visual appearance of the borders of the tooltips.
exToolTipBackColor	65	Specifies the tooltip's background color.
exToolTipForeColor	66	Specifies the tooltip's foreground color.
exCheckBoxState0	70	Specifies the visual appearance for the check box in 0 state (unchecked).
exCheckBoxState1	71	Specifies the visual appearance for the check box in 1 state (checked).
exCheckBoxState2	72	Specifies the visual appearance for the check box in 2 state (partial, not used).
exRadioButtonState0	73	Specifies the visual appearance for the radio button in 0 state (unchecked).
exRadioButtonState1	74	Specifies the visual appearance for the radio button in 1 state (checked).
exMenuFlatLineColor	100	Specifies the color to show the vertical line on flat appearance.
exMenuScrollBackColor	101	Specifies the background color to show the menu's scroll bars.
exMenuSelBorderColor	102	Specifies the color to show the frame around the selected item.
exMenuSeparatorItem	103	Specifies the color to show the separator item.
exMenuItemButton	104	Specifies the visual appearance for an item, when the Appearance property is Button.
		Specifies the visual appearance/solid color of the frame around the grouping items, when its group includes a single item. The GroupPopup property specifies the way the item's submenu are grouped.

Use the exGroupPopupFrameSingle, exGroupPopupFrameHStart, exGroupPopupFrameHIntermediate and exGroupPopupFrameHEnd to specify a different visual appearance for the frame around grouping items (horizontally).

exGroupPopupFrameSingle 105

The following screen shot shows the grouping items with an EBN frame:



which has been created using the following 4 EBNs (exGroupPopupFrameSingle, exGroupPopupFrameHStart, exGroupPopupFrameHIntermediate, exGroupPopupFrameHEnd):



exGroupPopupFrameHStart 106

Specifies the visual appearance/solid color of the frame around the first item (horizontally arranged), when the its group includes more items. The [GroupPopup](#) property specifies the way the item's submenu are grouped. Use the exGroupPopupFrameSingle, exGroupPopupFrameHStart, exGroupPopupFrameHIntermediate and exGroupPopupFrameHEnd to specify a different visual appearance for the frame around grouping items (horizontally).

exGroupPopupFrameHIntermediate 107

Specifies the visual appearance/solid color of the frame around an intermediate item (not start or end item, horizontally arranged), when the its group includes more items. The [GroupPopup](#) property specifies the way the item's submenu are grouped. Use the exGroupPopupFrameSingle, exGroupPopupFrameHStart, exGroupPopupFrameHIntermediate and exGroupPopupFrameHEnd to specify a different visual appearance for the frame around grouping items (horizontally).

exGroupPopupFrameHEnd	108	Specifies the visual appearance/solid color of the frame around the last item (not start or intermediate item, horizontally arranged), when the its group includes more items. The GroupPopup property specifies the way the item's submenu are grouped. Use the exGroupPopupFrameSingle, exGroupPopupFrameHStart, exGroupPopupFrameHIntermediate and exGroupPopupFrameHEnd to specify a different visual appearance for the frame around grouping items (horizontally).
-----------------------	-----	--

exGroupPopupFrameSolid	109	Specifies the solid color of the frame around the grouping items.
------------------------	-----	---

exMenuHotBackColor	110	Specifies the visual appearance/color to show the item from the cursor.
--------------------	-----	---

exMenuHotForeColor	111	Specifies the foreground color to show the item from the cursor.
--------------------	-----	--

exMenuSelHotBackColor	112	Specifies the visual appearance/color to show the selected item from the cursor.
-----------------------	-----	--

exMenuSelHotForeColor	113	Specifies the foreground color to show the selected item from the cursor.
-----------------------	-----	---

exMenuSeparatorSelectButton	114	Specifies the visual appearance/color to show the separator between select and drop down button.
-----------------------------	-----	--

exMenuSeparatorSelectButtonBottom	115	Specifies the visual appearance/color to show the separator between select and drop down button (show the drop-down button to the bottom).
-----------------------------------	-----	--

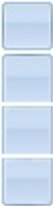
Specifies the visual appearance/solid color of the frame around the first item (vertically arranged), when the its group includes more items. The [GroupPopup](#) property specifies the way the item's submenu are grouped. Use the exGroupPopupFrameVStart, exGroupPopupFrameVIntermediate and exGroupPopupFrameVEnd to specify a different visual appearance for the frame around grouping items (vertically).

The following screen shot shows the grouping items with an EBN frame:

exGroupPopupFrameVStart 116



which has been created using the following 4 EBNs (exGroupPopupFrameSingle, exGroupPopupFrameVStart, exGroupPopupFrameVIntermediate, exGroupPopupFrameVEnd):



exGroupPopupFrameVIntermediate 117

Specifies the visual appearance/solid color of the frame around an intermediate item (not start or end item, vertically arranged), when the its group includes more items. The [GroupPopup](#) property specifies the way the item's submenu are grouped. Use the exGroupPopupFrameVStart, exGroupPopupFrameVIntermediate and exGroupPopupFrameVEnd to specify a different visual appearance for the frame around grouping items (vertically).

exGroupPopupFrameVEnd 118

Specifies the visual appearance/solid color of the frame around the last item (not intermediate or end item, vertically arranged), when the its group includes more items. The [GroupPopup](#) property specifies the way the item's submenu are grouped. Use the exGroupPopupFrameVStart, exGroupPopupFrameVIntermediate and exGroupPopupFrameVEnd to specify a different visual appearance for the frame around grouping items (vertically).

exMenuHotBorderColor 139

Specifies the color to show the frame around around the item from the cursor.

exMenuSelHotBorderColor 140

Specifies the color to show the frame around around the selected item from the cursor.

Specifies the visual appearance of the shortcut keys. The [Shortcut](#) property specifies the key

exShortcutKeyAppearance	143	combination that the user can press to select the item quickly.
exShortcutKeyBackColor	144	Specifies the shortcut keys' background color. The Shortcut property specifies the key combination that the user can press to select the item quickly.
exShortcutKeyForeColor	145	Specifies the shortcut keys' foreground color. The Shortcut property specifies the key combination that the user can press to select the item quickly.

constants CloseOnClickEnum

The CloseOnClickEnum type specifies when the user can close the popup menu. The [CloseOnClick](#) property specifies how the popup menu is closed when user clicks an item. The CloseOnClickEnum type supports the following values:

Name	Value	Description
exCloseOnClick	0	The popup menu is closing when the user clicks an item.
exCloseOnDbClick	1	The popup menu is closing when the user double clicks an item.
exCloseOnClickOutside	2	The popup menu is closing when the user clicks outside of the menu.
exCloseOnNonClickable	3	<p>The popup menu is closing when the user clicks a non-clickable item (regular items).</p> <p>Here's the list of clickable items:</p> <ul style="list-style-type: none">• separator items• item that hosts a sub-menu (popup item)• disabled item• check or radio items <p>For instance, clicking a check-box item will makes the check box to change its state instead closing the popup menu.</p>

constants CloseOnEnum

The CloseOnEnum type specifies how an item that contains an ActiveX inside is close. The [CloseOn](#) property indicates how the user closes the popup menu when an inside ActiveX control is clicked. The CloseOnEnum type supports the following values:

Name	Value	Description
exUser	0	The user is responsible for closing the popup menu.
exLButtonDown	513	The popup menu is closed when user presses the left mouse button over the ActiveX control.
exLButtonUp	514	The popup menu is closed when user releases the left mouse button over the ActiveX control.
exLButtonDblClk	515	The popup menu is closed when user double click the ActiveX control.
exRButtonDown	516	The popup menu is closed when user right clicks the ActiveX control.
exRButtonUp	517	The popup menu is closed when user releases the right mouse button over the ActiveX control.
exRButtonDblClk	518	The popup menu is closed when user double click the right mouse button in the ActiveX control.
exMButtonDown	519	The popup menu is closed when user clicks the middle mouse button in the ActiveX control.
exMButtonUp	520	The popup menu is closed when user releases the middle mouse button in the ActiveX control.
exMButtonDblClk	521	The popup menu is closed when user double clicks the middle mouse button in the ActiveX control.
exClick	61441	The popup menu is closed when user presses any of the mouse buttons in the ActiveX control.
exDbClick	61442	The popup menu is closed when user double clicks any of the mouse buttons in the ActiveX control.

constants EditBorderEnum

Specifies the type of the border around the edit control inside the item. Use the [AllowEdit](#) property to assign a single edit control to an item. Use the [EditBorder](#) property to specify the type of the border for the edit control inside the item.

Name	Value	Description
exEditBorderNone	0	No border.
exEditBorderInset	-1	Inset border.
exEditBorderSingle	1	Single border.

constants EditOptionEnum

The EditOptionEnum type specifies different options to be set / get for giving editor. The [EditOption](#) property specifies different options to be used for a specified edit field. The [AllowEdit](#) property associates an editor to the current item. The [EditCaption](#) property specifies the value to show in the edit field. The [EditWidth](#) property specifies the size/width of the edit field inside the item. The [EditBorder](#) property specifies whether the edit shows a border around it. The control fires the [EditChange](#) event when the user changes the edit's caption.

The control supports the following options:

Name	Value	Description
exEditMinValue	1	<p>Specifies the minimum value for the item's edit field. By default, the exEditMinValue option is 0. This option is valid for editors like: exItemEditSlider, exItemEditScrollBar</p> <p>(long expression)</p>
exEditMaxValue	2	<p>Specifies the maximum value for the item's edit field. By default, the exEditMinValue option is 100. This option is valid for editors like: exItemEditSlider, exItemEditScrollBar</p> <p>(long expression)</p>
exEditTickStyle	3	<p>Specifies where the ticks appears on the edit field. By default, the exEditTickStyle option is 1. The value of this option could be one of the following:</p> <ul style="list-style-type: none">• 0 (exBottomRight), The ticks are displayed on the bottom/right side• 1 (exTopLeft), The ticks are displayed on the top/left side• 2 (exBoth), The ticks are displayed on the both side• 3 (exNoTicks), No ticks are displayed <p>This option is valid for editors like: exItemEditSlider</p> <p>(long expression)</p>

exEditTickFrequency

4

Indicates the ratio of ticks on the control. By default, the exEditTickFrequency option is 10. This option is valid for editors like: exItemEditSlider (long expression)

Specifies the expression that determines the HTML labels to be shown on ticks.



For instance:

- "value", shows the values for each tick.
- "(value=current ? '<fgcolor=FF0000>' : ") + value", shows the current slider's position with a different color and font.
- "value = current ? value : """, shows the value for the current tick only.
- "(value = current ? '' : ") + (value array 'ab bc cd de ef fg gh hi ij jk kl' split ' ')" displays different captions for slider's values.

The option supports the following keywords:

- **value** gets the slider's position to be displayed
- **current** gets the current slider's value.
- **vmin** gets the slider's minimum value.
- **vmax** gets the slider's maximum value.
- **smin** gets the slider's selection minimum value.
- **smax** gets the slider's selection maximum value.

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** (and operator), priority 1

The supported binary boolean operators, all these with the same priority 0, are :

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

The supported 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 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:'

- ***in*** (*include operator*), specifies whether an element is found in a set of constant elements. The *in* operator returns -1 (True) if the element is found, else 0 (false) is retrieved. The syntax for *in* operator is

"expression in (c1,c2,c3,...cn)"

, where the c1, c2, ... are constant elements.

The constant elements could be numeric, date or string expressions. For instance the *"value*

in (11,22,33,44,13)" is equivalent with "

(expression = 11) or (expression = 22) or

(expression = 33) or (expression = 44) or

(expression = 13)". The *in* operator is not a

time consuming as the equivalent *or* version is,

so when you have large number of constant

elements it is recommended using the *in*

operator. Shortly, if the collection of elements

has 1000 elements the *in* operator could take

up to 8 operations in order to find if an element

fits the set, else if the *or* statement is used, it

could take up to 1000 operations to check, so

by far, the *in* operator could save time on

finding elements within a collection.

- ***switch*** (*switch operator*), returns the value being found in the collection, or a predefined value if the element is not found (default). The

syntax for *switch* operator is

"expression switch (default,c1,c2,c3,...,cn)"

, where the c1, c2, ... are constant elements, and the default is a constant element being returned when the element is not found in the collection. The constant elements could be numeric, date or string expressions. The equivalent syntax is "%0 = c 1 ? c 1 : (%0 = c 2 ? c 2 : (... ? . : default))". The *switch* operator is very similar with the *in* operator excepts that the first element in the switch is always returned by the statement if the element is not found, while the returned value is the value itself instead -1. For instance, the "%0 switch ('not found',1,4,7,9,11)" gets 1, 4, 7, 9 or 11, or 'not found' for any other value. As the *in* operator the *switch* operator uses binary searches for fitting the element, so it is quicker than iif (immediate if operator) alternative.

- **case()** (*case operator*) returns and executes one of n expressions, depending on the evaluation of the expression (IIF - immediate IF operator is a binary case() operator). The syntax for case() operator is:

"expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)"

If the default part is missing, the case() operator returns the value of the expression if it is not found in the collection of cases (c1, c2, ...). For instance, if the value of expression is not any of c1, c2, the default_expression is executed and returned. If the value of the expression is c1, then the case() operator executes and returns the expression1. The default, c1, c2, c3, ... must be constant elements as numbers, dates or strings. For instance, the "date(shortdate(value)) case

(default:0 ; #1/1/2002#:1 ; #2/1/2002#:1; #4/1/2002#:1; #5/1/2002#:1)" indicates that only #1/1/2002#, #2/1/2002#, #4/1/2002# and #5/1/2002# dates returns 1, since the others returns 0. For instance the following sample specifies the hour being non-working for specified dates: "date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)" statement indicates the working hours for dates as follows:

- - #4/1/2009#, from hours 06:00 AM to 12:00 PM
 - #4/5/2009#, from hours 07:00 AM to 10:00 AM and hours 03:00PM, 04:00PM, 06:00PM and 10:00PM
 - #5/1/2009#, from hours 12:00 AM to 08:00 AM

The *in*, *switch* and *case()* use binary search to look for elements so they are faster than using *if* and *or* expressions.

Obviously, the priority of the operations inside the expression is determined by () parenthesis and the priority for each operator.

The supported conversion unary operators are:

- **type** (unary operator) retrieves the type of the object. For instance `type(%0) = 8` specifies the cells that contains string values.

Here's few predefined types:

- 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
- **str** (unary operator) converts the expression to a string
- **dbl** (unary operator) converts the expression to a number
- **date** (unary operator) converts the expression to a date, based on your regional settings
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS.

exEditTickLabel

5

Other known operators for numbers are:

- **int** (unary operator) retrieves the integer part of the number
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2
- 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*' 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

Other known operators for strings are:

- **len** (unary operator) retrieves the number of characters in the string
- **lower** (unary operator) returns a string expression in lowercase letters
- **upper** (unary operator) returns a string expression in uppercase letters
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names
- **ltrim** (unary operator) removes spaces on the left side of a string
- **rtrim** (unary operator) removes spaces on the right side of a string
- **trim** (unary operator) removes spaces on both sides of a string
- **startswith** (binary operator) specifies whether a string starts with specified string
- **endwith** (binary operator) specifies whether a

- string ends with specified string
- **contains** (binary operator) specifies whether a string contains another specified string
- **left** (binary operator) retrieves the left part of the string
- **right** (binary operator) retrieves the right part of the string
- a **mid** b (binary operator) retrieves the middle part of the string a starting from b (1 means first position, and so on)
- a **count** b (binary operator) retrieves the number of occurrences of the b in a
- a **replace** b **with** c (double binary operator) replaces in a the b with c, and gets the result.
- a **split** b, 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

Other known operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel.
- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance the timeF(1:23 PM) returns "13:23:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel.
- **shortdateF** (unary operator) formats a date as a date string using the "MM/DD/YYYY" format. For instance the shortdateF(December 31, 1971 11:00 AM) returns "12/31/1971".
- **dateF** (unary operator) converts the date expression to a string expression in "MM/DD/YYYY HH:MM:SS" format.
- **longdate** (unary operator) formats a date as a date string using the long date format, as specified in the control's panel.
- **year** (unary operator) retrieves the year of the date (100,...,9999)

- **month** (unary operator) retrieves the month of the date (1, 2,...,12)
- **day** (unary operator) retrieves the day of the date (1, 2,...,31)
- **yearday** (unary operator) retrieves the number of the day in the year, or the days since January 1st (0, 1,...,365)
- **weekday** (unary operator) retrieves the number of days since Sunday (0 - Sunday, 1 - Monday,..., 6 - Saturday)
- **hour** (unary operator) retrieves the hour of the date (0, 1, ..., 23)
- **min** (unary operator) retrieves the minute of the date (0, 1, ..., 59)
- **sec** (unary operator) retrieves the second of the date (0, 1, ..., 59)

The The <VALUE> of [ticklabel] option can display labels using the following built-in HTML tags:

- **** displays the text in **bold**.
- **<i></i>** displays the text in *italics*.
- **<u></u>** underlines the text.
- **<s></s>** Strike-through text
- **** displays portions of text with a different font and/or different size. For instance, the bit draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, bit displays the bit text using the current font, but with a different size.
- **<fgcolor=RRGGBB></fgcolor>** displays text with a specified **foreground** color. The RR, GG or BB should be hexa values and indicates red, green and blue values.
- **<bgcolor=RRGGBB></bgcolor>** displays text with a specified **background** color. The RR, GG or BB should be hexa values and indicates red, green and blue values.
- **
** a forced line-break
- **<solidline>** The next line shows a solid-line on

top/bottom side. If has no effect for a single line caption.

- **<dotline>** The next line shows a dot-line on top/bottom side. If has no effect for a single line caption.
- **<upline>** The next line shows a solid/dot-line on top side. If has no effect for a single line caption.
- **<r>** Right aligns the text
- **<c>** Centers the text
- **number[:width]** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **key[:width]** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&**; (&), **<**; (<), **>**; (>), **&qout;** (") and **&#number** (the character with specified code), For instance, the **€** displays the EUR character, in UNICODE configuration. The **&** ampersand is only recognized as markup when it is followed by a known letter or a # character and a digit. For instance if you want to display

`bold` in HTML caption you can use
`bold`;

(string expression)

exEditSmallChange

6

The amount by which the edit's position changes when the user presses an arrow key. By default, the exEditSmallChange option is 1. This option is valid for editors like: exItemEditSlider, exItemEditScrollBar

(long expression)

exEditLargeChange

7

The amount by which the edit's position changes when the user presses an CTRL + arrow key. By default, the exEditLargeChange option is 5. This option is valid for editors like: exItemEditSlider, exItemEditScrollBar

(long expression)

Specifies the expression that determines the HTML tooltip to be shown when the item's value is changed. By default, the exEditChangeToolTip option is "value". This option is valid for editors like: exItemEditSlider, exItemEditScrollBar

The option supports the following keywords:

- **value** gets the slider's position to be displayed

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 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 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:'

- ***in*** (*include operator*), specifies whether an element is found in a set of constant elements. The *in* operator returns -1 (True) if the element is found, else 0 (false) is retrieved. The syntax for *in* operator is

"expression in (c1,c2,c3,...cn) "

, where the c1, c2, ... are constant elements.

The constant elements could be numeric, date or string expressions. For instance the *"value in (11,22,33,44,13)"* is equivalent with "

(expression = 11) or (expression = 22) or

(expression = 33) or (expression = 44) or

(expression = 13)". The *in* operator is not a

time consuming as the equivalent *or* version is,

so when you have large number of constant

elements it is recommended using the *in*

operator. Shortly, if the collection of elements

has 1000 elements the *in* operator could take

up to 8 operations in order to find if an element

fits the set, else if the *or* statement is used, it

could take up to 1000 operations to check, so

by far, the *in* operator could save time on

finding elements within a collection.

- ***switch*** (*switch operator*), returns the value being found in the collection, or a predefined value if the element is not found (default). The syntax for *switch* operator is

"expression switch (default,c1,c2,c3,...,cn) "

, where the *c1*, *c2*, ... are constant elements, and the default is a constant element being returned when the element is not found in the collection. The constant elements could be numeric, date or string expressions. The equivalent syntax is "%0 = c 1 ? c 1 : (%0 = c 2 ? c 2 : (... ? . : default))". The *switch* operator is very similar with the *in* operator excepts that the first element in the switch is always returned by the statement if the element is not found, while the returned value is the value itself instead -1. For instance, the "%0 switch ('not found',1,4,7,9,11)" gets 1, 4, 7, 9 or 11, or 'not found' for any other value. As the *in* operator the *switch* operator uses binary searches for fitting the element, so it is quicker than *iif* (immediate if operator) alternative.

- **case()** (*case operator*) returns and executes one of *n* expressions, depending on the evaluation of the expression (*IIF* - immediate IF operator is a binary *case()* operator). The syntax for *case()* operator is:

"expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)"

If the default part is missing, the *case()* operator returns the value of the expression if it is not found in the collection of cases (*c1*, *c2*, ...). For instance, if the value of expression is not any of *c1*, *c2*, the *default_expression* is executed and returned. If the value of the expression is *c1*, then the *case()* operator executes and returns the *expression1*. The *default*, *c1*, *c2*, *c3*, ... must be constant elements as numbers, dates or strings. For instance, the "*date(shortdate(value)) case (default:0 ; #1/1/2002#:1 ; #2/1/2002#:1 ; #4/1/2002#:1 ; #5/1/2002#:1)*" indicates that only *#1/1/2002#*, *#2/1/2002#*, *#4/1/2002#* and *#5/1/2002#* dates returns 1, since the

others returns 0. For instance the following sample specifies the hour being non-working for specified dates: "*date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)*" statement indicates the working hours for dates as follows:

- - #4/1/2009#, from hours 06:00 AM to 12:00 PM
 - #4/5/2009#, from hours 07:00 AM to 10:00 AM and hours 03:00PM, 04:00PM, 06:00PM and 10:00PM
 - #5/1/2009#, from hours 12:00 AM to 08:00 AM

The *in*, *switch* and *case()* use binary search to look for elements so they are faster than using *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. For instance *type(%0) = 8* specifies the cells that contains string values.

Here's few predefined types:

- 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
- **str** (unary operator) converts the expression to a string
- **dbl** (unary operator) converts the expression to a number
- **date** (unary operator) converts the expression to a date, based on your regional settings
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS.

Other known operators for numbers are:

- **int** (unary operator) retrieves the integer part of the number
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2
- 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*' 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

Other known operators for strings are:

- **len** (unary operator) retrieves the number of characters in the string
- **lower** (unary operator) returns a string expression in lowercase letters
- **upper** (unary operator) returns a string expression in uppercase letters
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names
- **ltrim** (unary operator) removes spaces on the left side of a string
- **rtrim** (unary operator) removes spaces on the right side of a string
- **trim** (unary operator) removes spaces on both sides of a string
- **startswith** (binary operator) specifies whether a string starts with specified string
- **endwith** (binary operator) specifies whether a string ends with specified string
- **contains** (binary operator) specifies whether a string contains another specified string
- **left** (binary operator) retrieves the left part of

the string

- **right** (binary operator) retrieves the right part of the string
- a **mid** b (binary operator) retrieves the middle part of the string a starting from b (1 means first position, and so on)
- a **count** b (binary operator) retrieves the number of occurrences of the b in a
- a **replace** b **with** c (double binary operator) replaces in a the b with c, and gets the result.
- a **split** b, 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

Other known operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel.
- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance the timeF(1:23 PM) returns "13:23:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel.
- **shortdateF** (unary operator) formats a date as a date string using the "MM/DD/YYYY" format. For instance the shortdateF(December 31, 1971 11:00 AM) returns "12/31/1971".
- **dateF** (unary operator) converts the date expression to a string expression in "MM/DD/YYYY HH:MM:SS" format.
- **longdate** (unary operator) formats a date as a date string using the long date format, as specified in the control's panel.
- **year** (unary operator) retrieves the year of the date (100,...,9999)
- **month** (unary operator) retrieves the month of the date (1, 2,...,12)
- **day** (unary operator) retrieves the day of the date (1, 2,...,31)

- **yearday** (unary operator) retrieves the number of the day in the year, or the days since January 1st (0, 1,...,365)
- **weekday** (unary operator) retrieves the number of days since Sunday (0 - Sunday, 1 - Monday,..., 6 - Saturday)
- **hour** (unary operator) retrieves the hour of the date (0, 1, ..., 23)
- **min** (unary operator) retrieves the minute of the date (0, 1, ..., 59)
- **sec** (unary operator) retrieves the second of the date (0, 1, ..., 59)

The The <VALUE> of [ticklabel] option can display labels using the following built-in HTML tags:

- **** displays the text in **bold**.
- **<i></i>** displays the text in *italics*.
- **<u></u>** underlines the text.
- **<s></s>** Strike-through text
- **** displays portions of text with a different font and/or different size. For instance, the **bit** draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, **bit** displays the bit text using the current font, but with a different size.
- **<fgcolor=RRGGBB></fgcolor>** displays text with a specified **foreground** color. The RR, GG or BB should be hexa values and indicates red, green and blue values.
- **<bgcolor=RRGGBB></bgcolor>** displays text with a specified **background** color. The RR, GG or BB should be hexa values and indicates red, green and blue values.
- **
** a forced line-break
- **<solidline>** The next line shows a solid-line on top/bottom side. If has no effect for a single line caption.
- **<dotline>** The next line shows a dot-line on top/bottom side. If has no effect for a single

line caption.

- **<upline>** The next line shows a solid/dot-line on top side. If has no effect for a single line caption.
- **<r>** Right aligns the text
- **<c>** Centers the text
- **number[:width]** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **key[:width]** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&**; (&), **<**; (<), **>**; (>), **&qout;** (") and **&#number** (the character with specified code), For instance, the **€** displays the EUR character, in UNICODE configuration. The **&** ampersand is only recognized as markup when it is followed by a known letter or a # character and a digit. For instance if you want to display **bold** in HTML caption you can use **bold**;

(string expression)

exEditMaskFloat

9

Specifies whether the mask field masks a floating point number. By default, the exEditMaskFloat is False. This flag is valid only for editors of exItemEditMask type.

(boolean expression)

exEditSpinStep

10

Specifies the step to advance when user clicks the editor's spin. By default, the exEditSpinStep is 1. This flag is valid only for editors of exItemEditSpin type.

(long expression)

constants GroupPopupMenu

The GroupPopupMenu type specifies whether the sub-menu of the current item is shown as grouped. The [GroupPopupMenu](#) property specifies whether the sub-menu of the current item is shown as grouped. The GroupPopupMenu type supports the following values:

Name	Value	Description
exNoGroupPopupMenu	0	No grouping is performed on the popup item.
exGroupPopupMenu	1	Groups and displays the sub-menu items on the current item, arranged from left to right (by default, the items are horizontally arranged).
exNoGroupPopupMenuFrame	2	Prevents showing the frame around each grouping item. If the exNoGroupPopupMenuFrame flag is not present, the control shows a frame around the grouping items. The frame around grouping items can be a solid box or EBN frames. The Background(exGroupPopupMenuFrameSolid) specifies the color to show the frames around grouping items, while the The Background(exGroupPopupMenuFrameSingle) , Background(exGroupPopupMenuFrameHStart) , Background(exGroupPopupMenuFrameHIntermediate) , Background(exGroupPopupMenuFrameHEnd) specifies the EBN to be shown for single element, starting element/item of the group, intermediate elements, and ending element/item. The exGroupPopupMenuFrameHStart , exGroupPopupMenuFrameHIntermediate and exGroupPopupMenuFrameHEnd are valid for horizontally grouping items (exGroupPopupMenuVertical flag is not present).
exGroupPopupMenuCenter	4	Shows the grouping popup aligned to the center of the current item.
exGroupPopupMenuRight	8	Shows the grouping popup aligned to the right of the current item.
exGroupPopupMenuEqualWidth	16	Shows the items that make the group of the same width.
exGroupPopupMenuEqualHeight	32	Shows the items that make the group of the same height.
		Forces the grouping items to show the solid frame

exGroupPopupFrameSolidBox64

(exGroupPopupFrameSolid) rather than EBN frame. The exGroupPopupFrameSolidBox flag can be combined with the exGroupPopupFrameThickBox to specify a ticker solid frame. This flag has no effect if the exNoGroupPopupFrame is present.

exGroupPopupFrameThickBox128

Specifies that the grouping items shows a ticker frame. The exGroupPopupFrameThickBox flag can be combined with the exGroupPopupFrameSolidBox to specify a ticker solid frame. This flag has no effect if the exNoGroupPopupFrame is present.

exGroupPopupVertical

256

Arranges vertically the items that compose the group (by default, the items are horizontally arranged). If the exNoGroupPopupFrame flag is not present, the control shows a frame around the grouping items. The frame around grouping items can be a solid box or EBN frames. The [Background](#)(exGroupPopupFrameSolid) specifies the color to show the frames around grouping items, while the The [Background](#)(exGroupPopupFrameSingle), [Background](#)(exGroupPopupFrameVStart), [Background](#)(exGroupPopupFrameVIntermediate), [Background](#)(exGroupPopupFrameVEnd) specifies the EBN to be shown for single element, starting element/item of the group, intermediate elements, and ending element/item. The exGroupPopupFrameVStart, exGroupPopupFrameVIntermediate and exGroupPopupFrameVEnd are valid for vertically grouping items (exGroupPopupVertical flag is present).

constants MenuItemTypeEnum

The MenuItemTypeEnum type specifies the type of Item objects to be collected using the [Get](#) method. The Get method can be used to get a collection / safe array of Item objects with a specified characteristics. For instance, you can collect the items of Edit type, or items that holds an Edit field inside. The MenuItemTypeEnum type supports the following values:

Name	Value	Description
exRegularMenuItem	0	Indicates a regular item. A regular item contains no sub-menus, no sub-control, no check box, no radio buttons and it is not a separator item.
exCheckBoxMenuItem	1	Indicates an item with a check box. The Check property specifies whether the item displays a check box.
exRadioButtonMenuItem	2	Indicates an item with a radio button. The Radio property specifies whether the item displays a radio button.
exSubMenuItem	3	Indicates a sub-menu item or an item that displays another menu.
exSeparatorMenuItem	4	Specifies a separator item. The SubMenu property gives access to the items collection being shown in the popup menu.
exControlItem	5	Indicates a sub-menu item that displays an ActiveX or a Window inside. The SubControl property gives access to the Control object being displayed.
exAllMenuItems	15	Indicates any type of item
exImageMenuItem	16	This flag can be combined with any other flag and value to specify the items that display icons using the Image property.
exDisplayTFIMenuItem	32	Reserved.
exEditMenuItem	64	This flag can be combined with any other flag and value to specify the items that display any Edit field inside. The EditCaption property specifies the caption to be shown in the item's edit field.
exDisabledMenuItem	128	This flag can be combined with any other flag and value to specify the disabled items, or items with the Enabled property on False.
		This flag can be combined with any other flag and

exUncheckedMenuItem	256	value to specify un-checked items, or items with the Checked property on False.
exCheckedMenuItem	512	This flag can be combined with any other flag and value to specify checked items, or items with the Checked property on True.
exPartialCheckedMenuItem	768	Reserved.

constants ModifierKeyEnum

The ModifierKeyEnum type specifies the modifies keys. The [ShortcutKeyPressedModifiers](#) property indicates the combination of modifier keys whose shortcut keys are currently visible. The ModifierKeyEnum type supports the following values:

Name	Value	Description
exNoModifier	0	(HEXA: 0x00000000) No modifier key.
exModifierShift	65536	(HEXA: 0x00010000) Indicates the SHIFT modifier key.
exModifierCtrl	131072	(HEXA: 0x00020000) Indicates the CTRL modifier key.
exModifierAlt	262144	(HEXA: 0x00040000) Indicates the ALT modifier key.
exModifierAny	-65536	(HEXA: 0xFFFF0000) Indicates any modifier key.

constants IncrementalSearchEnum

"In computing, incremental search, incremental find or real-time suggestions is a user interface interaction method to progressively search for and filter through text. As the user types text, one or more possible matches for the text are found and immediately presented to the user. ". The IncrementalSearchEnum type specifies how the control performs the incremental searching while user types characters. The [PopupIncrementalSearch](#) property the control's incremental search type.

Name	Value	Description
exNoIncrementalSearch	0	No incremental search is performed, when the user types characters.
exlSearchStartWith	1	Specifies that the control looks for objects that starts with typed characters, with highlighting the found result.
exlSearchContains	2	Specifies that the control looks for objects that contains typed characters, with highlighting the found result.
exlSearchFilterFor	16	The control displays just the items that match the typed characters.

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 ShortcutKeyAlignEnum

The ShortcutKeyAlignEnum type defines different type of alignments. The [ShortcutKeyAlignH](#) / [ShortcutKeyAlignV](#) property defines the alignment of the UI shortcut keys relative to the item that displays it. The ShortcutKeyAlignEnum type supports the following values:

Name	Value	Description
exShortcutKeyUpperLeft	0	Aligns the shortcut keys relative to upper-left point of the item.
exShortcutKeyUpperCenter	1	Aligns the shortcut keys relative to upper-point point of the item.
exShortcutKeyUpperRight	2	Aligns the shortcut keys relative to upper-right point of the item.
exShortcutKeyMiddleLeft	16	Aligns the shortcut keys relative to middle-left point of the item.
exShortcutKeyMiddleCenter	17	Aligns the shortcut keys relative to middle-center point of the item.
exShortcutKeyMiddleRight	18	Aligns the shortcut keys relative to middle-right point of the item.
exShortcutKeyLowerLeft	32	Aligns the shortcut keys relative to lower-left point of the item.
exShortcutKeyLowerCenter	33	Aligns the shortcut keys relative to lower-center point of the item.
exShortcutKeyLowerRight	34	Aligns the shortcut keys relative to lower-right point of the item.



constants ShortcutKeyVisibleEnum

The ShortcutKeyVisibleEnum type defines whether the shortcut key is visible or hidden. The [ShortcutKeyVisible](#) property gets or sets a value that specifies whether the control's shortcut keys are visible or hidden. The ShortcutKeyVisibleEnum type supports the following flags:

Name	Value	Description
exHideShortcutKeys	0	No visual appearance is shown for any of the shortcut keys.
exShowShortcutKeysPressOnly	1	The visual appearance of each available shortcut key is shown as soon as the user presses the modifier keys.
exShowShortcutKeysPressOnlyDelayed	2	The visual appearance of each available shortcut key is shown delayed as soon as the user presses the modifier keys.
exShowShortcutKeysToggle	3	The visual appearance of each available shortcut key is shown or hidden when the user releases the modifier keys.
exShowShortcutKeysToggleDelayed	4	The visual appearance of each available shortcut key is shown or hidden when the user releases the modifier keys, or as soon as the user keeps pressing the modifier keys.
exDisableShortcutKeys	16	All shortcut keys are disabled. This flag can be combined with any other.
exDisplayShortcutKeysAllAvailable	32	All available shortcut keys are shown. This flag can be combined with any other.
exLongerDelayShortcutKeys	64	Specifies a longer delay for Delayed modes. This flag can be combined with any other.
exAllowShortcutKeysIfNotShown	128	Indicates that the user can invoke any shortcut, even if it is not visible. This flag can be combined with any other.
exCloseShortcutKeysOnClick	256	Specifies to close the shortcut keys when the user click anywhere. This flag can be combined with any other.

constants ShowAsButtonEnum

The ShowAsButtonEnum type specifies the way a button is shown. The [ShowAsButton](#) property specifies whether the current item is shown a button or a select button. The ShowAsButtonEnum type supports the following values:

Name	Value	Description
exShowAsButtonNone	0	No button is associated with the current item.
exShowAsButton	1	<div>A button is associated with the current item. This flag can be combined with exShowAsButtonAutoSize flag.</div> <div></div>
exShowAsButtonAutoSize	2	The size of the button's caption fits the item's caption. This flag can be combined with any other flags.
exShowAsSelectButton	17	<div>A select button is associated with the current item (show the drop-down button to the right). The exShowAsSelectButton flag has effect only if the current item is a popup item, so it contains sub-items. The SubMenu property gives access to the item's sub menu, while it is a popup item. The Add(Caption,SubMenu) adds a popup item. The item's SubMenu is shown bellow the button, when user clicks the associated arrow. The popup being shown is a local popup, so clicking any item inside a local popup makes the popup itself to close including all its descendent sub-menus, without closing any ascendant sub-menus.</div> <div></div>

A select button is associated with the current item (show the drop-down button to the bottom). The exShowAsSelectButtonBottom flag has effect only if the current item is a popup item, so it contains sub-items. The [SubMenu](#) property gives access to the item's sub menu, while it is a popup item. The [Add](#)(Caption, SubMenu) adds a popup item (an

exShowAsSelectButtonBottom273

item that contains sub-items). The item's [SubMenu](#) is shown bellow the button, when user clicks the associated arrow. The popup being shown is a local popup, so clicking any item inside a local popup makes the popup itself to close including all its descendent sub-menus, without closing any ascendant sub-menus.



constants ShowCheckedAsSelectedEnum

The [ShowCheckedAsSelected](#) property specifies whether the checked items (all) shows as selected. The [ShowCheckedAsSelected](#) property of the Item object specifies whether the individual checked item is shown as selected. The ShowCheckedAsSelectedEnum type supports the following values.

Name	Value	Description
exDisplayItemCheckDefault	0	No highlighting is applied to item.
exDisplayItemCheckHighlight	-1	Highlights or un-highlights the checked/unchecked item, but still the check/radio buttons are shown.
exDisplayItemHighlight	1	Highlights or un-highlights the checked/unchecked item, but check/radio buttons are hidden.
exDisplayItemCheckInherit	2	Inherits the value of the control's ShowCheckedAsSelected property.

constants ShowPopupAlignEnum

The ShowPopupAlign type specifies the position to show the drop down popup menu. The [ShowPopupAlign](#) property specifies how the item's sub-menu is aligned relative to the parent item. The ShowPopupAlign type supports the following values:

Name	Value	Description
exShowPopupAlignNone	0	The sub-menu/popup is shown up-left aligned relative to the parent item.
exShowPopupAlignDown	1	The sub-menu/popup is shown down-aligned relative to the parent item.
exShowPopupAlignRight	2	The sub-menu/popup is shown right-aligned relative to the parent item.

constants ShowPopupArrowEnum

The ShowPopupArrowEnum type specifies how the arrow of an item that displays a sub-menu is shown. The [ShowPopupArrow](#) specifies how the arrow of an item that displays a sub-menu is shown. The ShowPopupArrowEnum supports the following values:

Name	Value	Description
exPopupArrowNone	0	No arrow is shown.
exShowPopupArrowLight	1	The item shows a light arrow when it displays a sub-menu.
exShowPopupArrowDark	2	The item shows a dark arrow when it displays a sub-menu.

constants ShowPopupEffectEnum

The ShowPopupEffectEnum value indicates the effect to be shown, when the user clicks an item with a sub-menu associated. The [ShowPopupEffect](#) property indicates the effect to be applied when the popup menu is shown. The ShowPopupEffectEnum type supports the following values:

Name	Value	Description
exShowPopupDirect	0	The popup menu is shown directly, with no effect.
exShowPopupScroll	-1	The popup menu is scrolling before showing.
exShowPopupLightUp	1	The popup menu is lightning up before showing.

constants SubMenuSortOrderEnum

The SubMenuSortOrderEnum type specifies the way the submenu displays the items. The [SortOrder](#) property specifies the sort order to display the items in the sub menu. The SubMenuSortOrderEnum type supports the following values:

Name	Value	Description
exSubMenuUnsorted	0	No sort is applied when the submenu is displayed.
exSubMenuAscending	1	The items in the submenu are alphabetically displayed in ascending order.
exSubMenuDescending	2	The items in the submenu are alphabetically displayed in descending order.
exSubMenuReverse	3	The items in the submenu are displayed in reverse order.

constants ItemTypeEnum

The ItemTypeEnum type specifies the type of items to be added to the control. The ItemType parameter of the [Add](#) method specifies the type of the item to be added to the [Items](#) collection. The ItemTypeEnum type supports the following values:

Name	Value	Description
Regular	0	Specifies a regular item, with no sub menu.
Separator	1	Specifies a separator item. The Background(exMenuSeparatorItem) property specifies the visual appearance of the separator items.
SubMenu	2	Specifies a sub menu. The SubMenu property gets a collection of Item objects to be displayed on the sub-menu.
SubControl	3	Specifies a popup menu that hosts an ActiveX control or a Window. The SubControl property gets access to the Control object that holds information about the inside ActiveX or Window hosted by the item

constants UVisualThemeEnum

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

Name	Value	Description
exNoVisualTheme	0	exNoVisualTheme
exDefaultVisualTheme	16777215	exDefaultVisualTheme
exHeaderVisualTheme	1	exHeaderVisualTheme
exFilterBarVisualTheme	2	exFilterBarVisualTheme
exButtonsVisualTheme	4	exButtonsVisualTheme
exCalendarVisualTheme	8	exCalendarVisualTheme
exSliderVisualTheme	16	exSliderVisualTheme
exSpinVisualTheme	32	exSpinVisualTheme
exCheckBoxVisualTheme	64	exCheckBoxVisualTheme
exProgressVisualTheme	128	exProgressVisualTheme
exCalculatorVisualTheme	256	exCalculatorVisualTheme

Appearance object

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

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

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

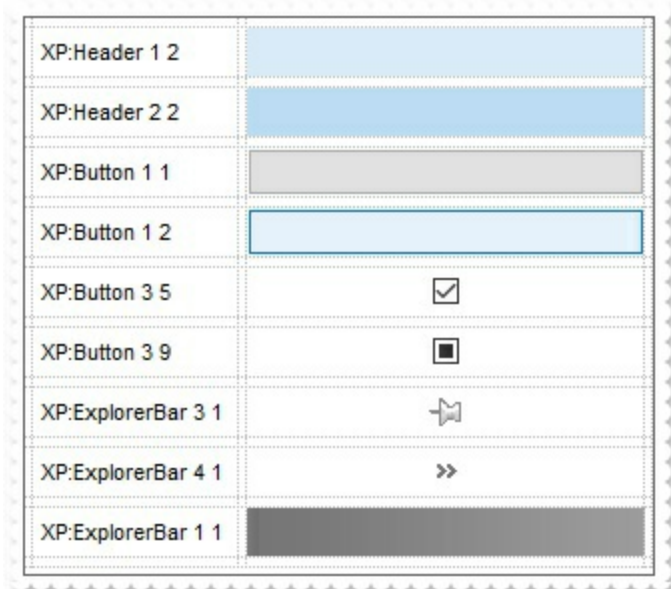
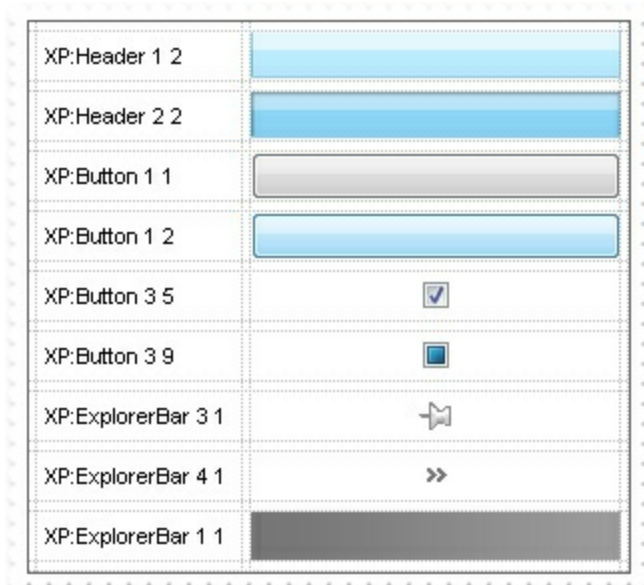
Adds or replaces a skin object to the control.

Type	Description
ID as Long	<p>A Long expression that indicates the index of the skin being added or replaced. The value must be between 1 and 126, so Appearance collection should holds no more than 126 elements.</p> <p>The Skin parameter of the Add method can a STRING as explained bellow, a BYTE[] / safe arrays of VT_I1 or VT_UI1 expression that indicates the content of the EBN file. You can use the BYTE[] / safe arrays of VT_I1 or VT_UI1 option when using the EBN file directly in the resources of the project. For instance, the VB6 provides the LoadResData to get the safe array o bytes for specified resource, while in VB/.NET or C# the internal class Resources provides definitions for all files being inserted. (ResourceManager.GetObject("ebn", resourceCulture))</p> <p>If the Skin parameter points to a string expression, it can be one of the following:</p> <ul style="list-style-type: none">• A path to the skin file (*.EBN). The ExButton component or ExEBN tool can be used to create, view or edit EBN files. For instance, "C:\Program Files\Exontrol\ExButton\Sample\EBN\MSOffice-Ribbon\msor_frameh.ebn"• A BASE64 encoded string that holds the skin file (*.EBN). Use the ExImages tool to build BASE 64 encoded strings of the skin file (*.EBN). The BASE64 encoded string starts with "gBFLBCJw..."• An Windows XP theme part, if the Skin parameter starts with "XP:". Use this option, to display any UI element of the Current Windows XP Theme, on any part of the control. In this case, the syntax of the Skin parameter is: "XP:ClassName Part State" where the ClassName defines the window/control class name in the Windows XP Theme, the Part indicates a long expression that defines the part, and the State indicates the state of the part to be shown. All known values for window/class, part and start are defined at

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

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

Skin as Variant



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

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

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



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

Use the Add method to add or replace skins to the control. The skin method, in it's simplest form, uses a single graphic file (*.ebn) assigned to a part of the control. 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 [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.

Starting with **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
		GBS_NORMAL = 1
		GBS_DISABLED =
	BP_GROUPBOX = 4	PBS_NORMAL = 1
		= 2 PBS_PRESSED
		PBS_DISABLED =
	BP_PUSHBUTTON = 1	PBS_DEFAULTED :
		RBS_UNCHECKED
		1 RBS_UNCHECKE
		RBS_UNCHECKED
		= 3
		RBS_UNCHECKED
		= 4 RBS_CHECKED
		5 RBS_CHECKEDH
		RBS_CHECKEDPR
		RBS_CHECKEDDIS
	BP_RADIOBUTTON = 2	
	BP_USERBUTTON = 5	

CLOCK	CLP_TIME = 1	CLS_NORMAL = 1 CBXS_NORMAL = CBXS_HOT = 2 CBXS_PRESSED = CBXS_DISABLED :
COMBOBOX	CP_DROPDOWNBUTTON = 1	
EDIT	EP_CARET = 2 EP_EDITTEXT = 1	ETS_NORMAL = 1 2 ETS_SELECTED ETS_DISABLED = ETS_FOCUSED = ETS_READONLY = ETS_ASSIST = 7
EXPLORERBAR	EBP_HEADERBACKGROUND = 1 EBP_HEADERCLOSE = 2 EBP_HEADERPIN = 3 EBP_IEBARMENU = 4 EBP_NORMALGROUPBACKGROUND = 5 EBP_NORMALGROUPCOLLAPSE = 6 EBP_NORMALGROUPEXPAND = 7 EBP_NORMALGROUPHEAD = 8 EBP_SPECIALGROUPBACKGROUND = 9 EBP_SPECIALGROUPCOLLAPSE = 10 EBP_SPECIALGROUPEXPAND = 11	EBHC_NORMAL = EBHC_HOT = 2 EBHC_PRESSED = EBHP_NORMAL = EBHP_HOT = 2 EBHP_PRESSED = EBHP_SELECTED 4 EBHP_SELECTED EBHP_SELECTED 6 EBM_NORMAL = 1 = 2 EBM_PRESSE EBNGC_NORMAL : EBNGC_HOT = 2 EBNGC_PRESSED EBNGE_NORMAL : EBNGE_HOT = 2 EBNGE_PRESSED EBSGC_NORMAL : EBSGC_HOT = 2 EBSGC_PRESSED EBSGE_NORMAL : EBSGE_HOT = 2 EBSGE_PRESSED

EBP_SPECIALGROUPHEAD = 12

HEADER

HP_HEADERITEM = 1

HP_HEADERITEMLEFT = 2

HP_HEADERITEMRIGHT = 3

HP_HEADERSORTARROW = 4

HIS_NORMAL = 1 HIS_PRESSED = 2

HILS_NORMAL = 1 HILS_PRESSED = 2

HIRS_NORMAL = 1 HIRS_PRESSED = 2

HSAS_SORTEDUP HSAS_SORTEDDC

LISTVIEW

LVP_EMPTYTEXT = 5

LVP_LISTDETAIL = 3

LVP_LISTGROUP = 2

LVP_LISTITEM = 1

LVP_LISTSORTEDDETAIL = 4

LIS_NORMAL = 1 LIS_SELECTED = 2
LIS_DISABLED = 4 LIS_SELECTEDNO
5

MENU

MP_MENUBARDROPDOWN = 4

MP_MENUBARITEM = 3

MP_CHEVRON = 5

MP_MENUDROPDOWN = 2

MP_MENUITEM = 1

MP_SEPARATOR = 6

MS_NORMAL = 1 MS_SELECTED = 2
MS_DEMOTED = 3

MS_NORMAL = 1 MS_SELECTED = 2
MS_DEMOTED = 3

MS_NORMAL = 1 MS_SELECTED = 2
MS_DEMOTED = 3

MS_NORMAL = 1 MS_SELECTED = 2
MS_DEMOTED = 3

MS_NORMAL = 1 MS_SELECTED = 2
MS_DEMOTED = 3

MS_NORMAL = 1 MS_SELECTED = 2
MS_DEMOTED = 3

MDS_NORMAL = 1 MDS_PRESSED = 2
MDS_DISABLED = MDS_CHECKED =

MENUBAND

MDP_NEWAPPBUTTON = 1

		MDS_HOTCHECKE
	MDP_SEPERATOR = 2	
PAGE	PGRP_DOWN = 2	DNS_NORMAL = 1 = 2 DNS_PRESSE DNS_DISABLED = DNHZS_NORMAL = DNHZS_HOT = 2 DNHZS_PRESSED DNHZS_DISABLED
	PGRP_DOWNHORZ = 4	UPS_NORMAL = 1 = 2 UPS_PRESSE UPS_DISABLED = UPHZS_NORMAL = UPHZS_HOT = 2 UPHZS_PRESSED UPHZS_DISABLED
	PGRP_UP = 1	
	PGRP_UPHORZ = 3	
PROGRESS	PP_BAR = 1 PP_BARVERT = 2 PP_CHUNK = 3 PP_CHUNKVERT = 4	
REBAR	RP_BAND = 3 RP_CHEVRON = 4 RP_CHEVRONVERT = 5 RP_GRIPPER = 1 RP_GRIPPERVERT = 2	CHEVS_NORMAL = CHEVS_HOT = 2 CHEVS_PRESSED
		ABS_DOWNDISAB ABS_DOWNHOT, ABS_DOWNNORM ABS_DOWNPRES ABS_UPDISABLED ABS_UPHOT, ABS_UPNORMAL, ABS_UPPRESSED, ABS_LEFTDISAB ABS_LEFTHOT, ABS_LEFTNORMA ABS_LEFTPRESSE ABS_RIGHTDISAB
SCROLLBAR	SBP_ARROWBTN = 1	

SPIN

SBP_GRIPPERHORZ = 8
SBP_GRIPPERVERT = 9

SBP_LOWERTRACKHORZ = 4

SBP_LOWERTRACKVERT = 6

SBP_THUMBBTNHORZ = 2

SBP_THUMBBTNVERT = 3

SBP_UPPERTRACKHORZ = 5

SBP_UPPERTRACKVERT = 7

SBP_SIZEBOX = 10

SPNP_DOWN = 2

SPNP_DOWNHORZ = 4

SPNP_UP = 1

ABS_RIGHTHOT,
ABS_RIGHTNORM
ABS_RIGHTPRESSED

SCRBS_NORMAL :
SCRBS_HOT = 2
SCRBS_PRESSED
SCRBS_DISABLED

SCRBS_NORMAL :
SCRBS_HOT = 2
SCRBS_PRESSED
SCRBS_DISABLED

SCRBS_NORMAL :
SCRBS_HOT = 2
SCRBS_PRESSED
SCRBS_DISABLED

SCRBS_NORMAL :
SCRBS_HOT = 2
SCRBS_PRESSED
SCRBS_DISABLED

SCRBS_NORMAL :
SCRBS_HOT = 2
SCRBS_PRESSED
SCRBS_DISABLED

SCRBS_NORMAL :
SCRBS_HOT = 2
SCRBS_PRESSED
SCRBS_DISABLED

SZB_RIGHTALIGN
SZB_LEFTALIGN =

DNS_NORMAL = 1
= 2 DNS_PRESSED
DNS_DISABLED =

DNHZZ_NORMAL :
DNHZZ_HOT = 2
DNHZZ_PRESSED
DNHZZ_DISABLED

UPS_NORMAL = 1
= 2 UPS_PRESSED
UPS_DISABLED =

SPNP_UPHORZ = 3

UPHZS_NORMAL =
UPHZS_HOT = 2
UPHZS_PRESSED
UPHZS_DISABLED

STARTPANEL

SPP_LOGOFF = 8

SPLS_NORMAL =
SPLS_HOT = 2
SPLS_PRESSED =

SPP_LOGOFFBUTTONS = 9

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
2 TIS_SELECTED :
TIS_DISABLED = 4
TIS_FOCUSED = 5
TIBES_NORMAL =
TIBES_HOT = 2
TIBES_SELECTED
TIBES_DISABLED
TIBES_FOCUSED :
TILES_NORMAL =
TILES_HOT = 2
TILES_SELECTED
TILES_DISABLED :
TILES_FOCUSED :
TIRES_NORMAL =
TIRES_HOT = 2

TABP_TABITEMBOTHEDGE = 4

TABP_TABITEMLEFTEDGE = 2

TABP_TABITEMRIGHTEDGE = 3

TABP_TOPTABITEM = 5

TABP_TOPTABITEMBOTHEDGE = 8

TABP_TOPTABITEMLEFTEDGE = 6

TABP_TOPTABITEMRIGHTEDGE = 7

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

TIRES_SELECTED

TIRES_DISABLED

TIRES_FOCUSED

TTIS_NORMAL = 1

= 2 TTIS_SELECTED

TTIS_DISABLED =

TTIS_FOCUSED =

TTIBES_NORMAL :

TTIBES_HOT = 2

TTIBES_SELECTED

TTIBES_DISABLED

TTIBES_FOCUSED

TTILES_NORMAL :

TTILES_HOT = 2

TTILES_SELECTED

TTILES_DISABLED

TTILES_FOCUSED

TTIRES_NORMAL :

TTIRES_HOT = 2

TTIRES_SELECTED

TTIRES_DISABLED

TTIRES_FOCUSED

TS_NORMAL = 1 T

TS_PRESSED = 3

TS_DISABLED = 4

TS_CHECKED = 5

TS_HOTCHECKED

TS_NORMAL = 1 T

TS_PRESSED = 3

TP_DROPDOWNBUTTON = 2

TS_DISABLED = 4

TS_CHECKED = 5

TS_HOTCHECKED

TS_NORMAL = 1 T

TS_PRESSED = 3

TP_SPLITBUTTON = 3

TS_DISABLED = 4

TS_CHECKED = 5

TS_HOTCHECKED

TS_NORMAL = 1 T

TS_PRESSED = 3

TP_SPLITBUTTONDROPDOWN = 4

TS_DISABLED = 4

TS_CHECKED = 5

TS_HOTCHECKED

TS_NORMAL = 1 T

TS_PRESSED = 3

TP_SEPARATOR = 5

TS_DISABLED = 4

TS_CHECKED = 5

TS_HOTCHECKED

TS_NORMAL = 1 T

TS_PRESSED = 3

TP_SEPARATORVERT = 6

TS_DISABLED = 4

TS_CHECKED = 5

TS_HOTCHECKED

TTBS_NORMAL =

TTBS_LINK = 2

TTBS_NORMAL =

TTBS_LINK = 2

TTCS_NORMAL =

TTCS_HOT = 2

TTCS_PRESSED =

TTSS_NORMAL =

TTSS_LINK = 2

TTSS_NORMAL =

TTSS_LINK = 2

TUS_NORMAL = 1

2 TUS_PRESSED =

TUS_FOCUSED =

TUS_DISABLED =

TUBS_NORMAL =

TUBS_HOT = 2

TUBS_PRESSED =

TOOLTIP

TTP_BALLOON = 3

TTP_BALLOONTITLE = 4

TTP_CLOSE = 5

TTP_STANDARD = 1

TTP_STANDARDTITLE = 2

TRACKBAR

TKP_THUMB = 3

TKP_THUMBBOTTOM = 4

TKP_THUMBLEFT = 7

TKP_THUMBRIGHT = 8

TKP_THUMBTOP = 5

TKP_THUMBVERT = 6

TKP_TICS = 9

TKP_TICSVERT = 10

TKP_TRACK = 1

TKP_TRACKVERT = 2

TRAYNOTIFY

TNP_ANIMBACKGROUND = 2

TNP_BACKGROUND = 1

TREEVIEW

TVP_BRANCH = 3

TVP_GLYPH = 2

TVP_TREEITEM = 1

WINDOW

WP_CAPTION = 1

WP_CAPTIONSIZINGTEMPLATE = 30

TUBS_FOCUSED =
TUBS_DISABLED =
TUVLS_NORMAL =
TUVLS_HOT = 2
TUVLS_PRESSED
TUVLS_FOCUSED
TUVLS_DISABLED

TUVRN_NORMAL =
TUVRN_HOT = 2
TUVRN_PRESSED
TUVRN_FOCUSED
TUVRN_DISABLED

TUTS_NORMAL =
TUTS_HOT = 2
TUTS_PRESSED =
TUTS_FOCUSED =
TUTS_DISABLED =

TUVS_NORMAL =
TUVS_HOT = 2
TUVS_PRESSED =
TUVS_FOCUSED =
TUVS_DISABLED =

TSS_NORMAL = 1
TSVS_NORMAL =
TRS_NORMAL = 1
TRVS_NORMAL =

GLPS_CLOSED =
GLPS_OPENED =
TREIS_NORMAL =
TREIS_HOT = 2
TREIS_SELECTED
TREIS_DISABLED
TREIS_SELECTED
= 5

CS_ACTIVE = 1 CS
= 2 CS_DISABLED

WP_CLOSEBUTTON = 18

WP_DIALOG = 29

WP_FRAMEBOTTOM = 9

WP_FRAMEBOTTOMSIZINGTEMPLATE = 36

WP_FRAMELEFT = 7

WP_FRAMELEFTSIZINGTEMPLATE = 32

WP_FRAMERIGHT = 8

WP_FRAMERIGHTSIZINGTEMPLATE = 34

WP_HELPBUTTON = 23

WP_HORIZSCROLL = 25

WP_HORIZTHUMB = 26

WP_MAX_BUTTON

WP_MAXCAPTION = 5

WP_MDICLOSEBUTTON = 20

WP_MDIHELPBUTTON = 24

WP_MDIMINBUTTON = 16

CBS_NORMAL = 1
= 2 CBS_PUSHED
CBS_DISABLED =

FS_ACTIVE = 1 FS
= 2

FS_ACTIVE = 1 FS
= 2

FS_ACTIVE = 1 FS
= 2

HBS_NORMAL = 1
= 2 HBS_PUSHED
HBS_DISABLED =

HSS_NORMAL = 1
= 2 HSS_PUSHED
HSS_DISABLED =

HTS_NORMAL = 1
2 HTS_PUSHED =
HTS_DISABLED =

MAXBS_NORMAL
MAXBS_HOT = 2
MAXBS_PUSHED =
MAXBS_DISABLED

MXCS_ACTIVE = 1
MXCS_INACTIVE =
MXCS_DISABLED

CBS_NORMAL = 1
= 2 CBS_PUSHED
CBS_DISABLED =

HBS_NORMAL = 1
= 2 HBS_PUSHED
HBS_DISABLED =

MINBS_NORMAL =
MINBS_HOT = 2
MINBS_PUSHED =
MINBS_DISABLED

RBS_NORMAL = 1

WP_MDIRESTOREBUTTON = 22	= 2 RBS_PUSHED RBS_DISABLED = SBS_NORMAL = 1 = 2 SBS_PUSHED SBS_DISABLED = MINBS_NORMAL = MINBS_HOT = 2 MINBS_PUSHED = MINBS_DISABLED
WP_MDISYSBUTTON = 14	MNCS_ACTIVE = 1 MNCS_INACTIVE = MNCS_DISABLED RBS_NORMAL = 1 = 2 RBS_PUSHED RBS_DISABLED = CS_ACTIVE = 1 CS = 2 CS_DISABLED
WP_MINBUTTON = 15	CBS_NORMAL = 1 = 2 CBS_PUSHED CBS_DISABLED = FS_ACTIVE = 1 FS = 2
WP_MINCAPTION = 3	FS_ACTIVE = 1 FS = 2
WP_RESTOREBUTTON = 21	FS_ACTIVE = 1 FS = 2
WP_SMALLCAPTION = 2	HBS_NORMAL = 1 = 2 HBS_PUSHED HBS_DISABLED = MAXBS_NORMAL MAXBS_HOT = 2 MAXBS_PUSHED = MAXBS_DISABLED
WP_SMALLCAPTIONSIZINGTEMPLATE = 31	
WP_SMALLCLOSEBUTTON = 19	
WP_SMALLFRAMEBOTTOM = 12	
WP_SMALLFRAMEBOTTOMSIZINGTEMPLATE = 37	
WP_SMALLFRAMELEFT = 10	
WP_SMALLFRAMELEFTSIZINGTEMPLATE = 33	
WP_SMALLFRAMERIGHT = 11	
WP_SMALLFRAMERIGHTSIZINGTEMPLATE = 35	
WP_SMALLHELPBUTTON	
WP_SMALLMAXBUTTON	

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.

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.


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"
    .Property = &H1000000
End With
```

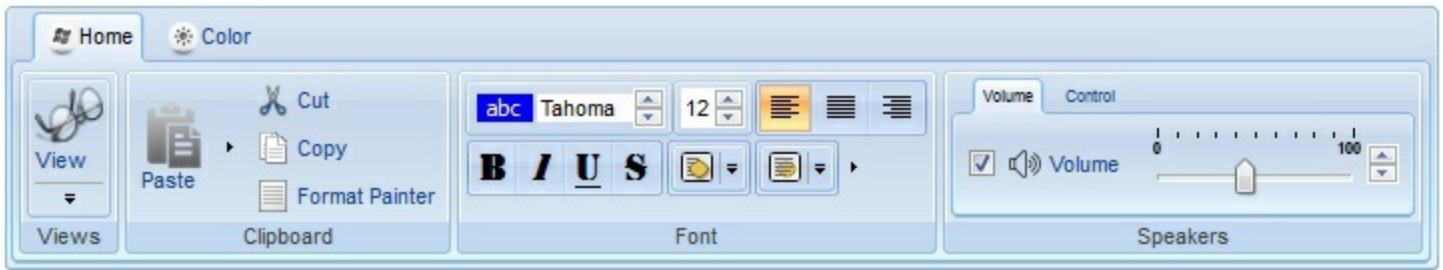
In the following screen shot the following objects displays the current EBN with a different color:

- "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:

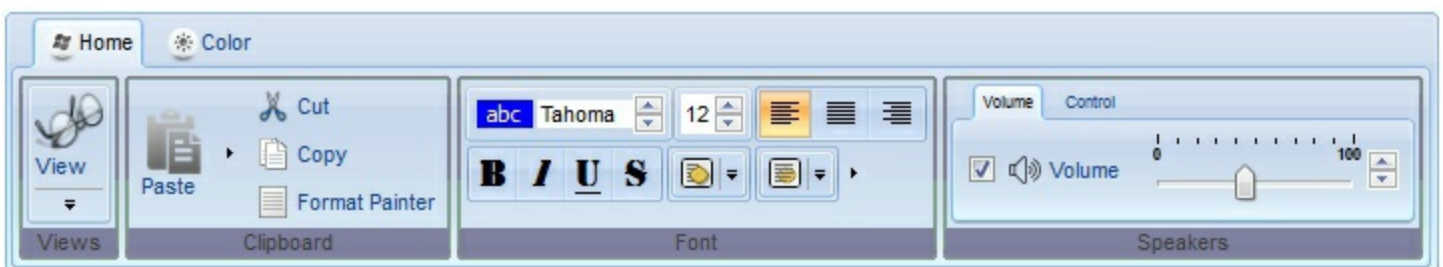
- **-3, no color is applied.** For instance, the Property = &H1FF0000 is displayed as would be .Property = &H1000000, so the 0xFF0000 color (Blue color) is ignored. You can use this option to allow the control displays the EBN colors or not.



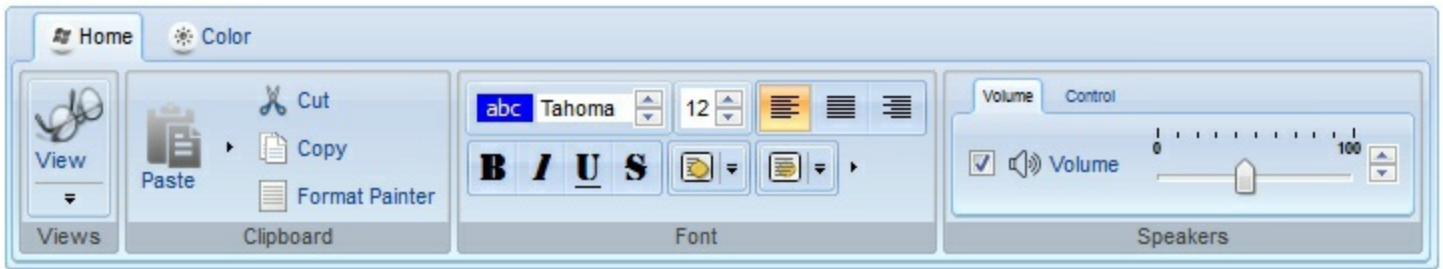
- **-2, OR-color scheme.** The color to be applied on the part of the control is a OR bit combination between the original EBN color and the specified color. For instance, the Property = &H1FF0000, applies the OR bit for the entire Blue channel, or in other words, it applies a less Blue to the part of 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 Property = &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 Property = &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 0x4000FFFF (25% Yellow, 0x40 or 64 in decimal is 25% from 256):



The following picture shows the control with the RenderType property on 0x8000FFFF (50% Yellow, 0x80 or 128 in decimal is 50% from 256):



The following picture shows the control with the RenderType property on 0xC000FFFF (75% Yellow, 0xC0 or 192 in decimal is 75% from 256):



The following picture shows the control with the `RenderType` property on `0xFF00FFFF` (100% Yellow, `0xFF` or 255 in decimal is 100% from 255):



Control object

The Control object holds properties to access the ActiveX or Window to be hosted by the item. The eXRibbon component can host any ActiveX control or an already created window. The Control object can be accessed through the [SubControl](#) property of the Item object. The Control object supports the following properties and methods:

Name	Description
CloseOn	Indicates when the control is closed.
ControlID	Specifies the control's identifier.
Create	Creates the component.
Height	Specifies the control's height.
LicenseKey	Specifies the control's runtime license key.
Object	Gets the object.
Width	Specifies the control's width.
Window	Specifies the handle of the window to be hosted.

property Control.CloseOn as CloseOnEnum

Indicates when the control is closed.

Type	Description
CloseOnEnum	A CloseOnEnum expression that specifies when the popup menu is closed once the user clicks the inside ActiveX control.

By default, the CloseOn property is exUser. In other words, the popup menu is not closed once the user clicks the ActiveX control.

property Control.ControlID as String

Specifies the control's identifier.

Type	Description
String	A string expression that indicates the control's identifier.

The ControlID and [LicenseKey](#) properties must be set before calling [Create](#) method. The Create method creates an ActiveX control given its identifier and its runtime license key, if required. A control identifier, or programmatic identifier, is a registry entry that can be associated with a CLSID. The format of a control identifier is <Vendor>.<Component>.<Version>, separated by periods and with no spaces, as in Word.Document.6.

For instance, the control's identifier for Microsoft Calendar Control is "MSCAL.Calendar", the control's identifier for Exontrol ExGrid Control is "Exontrol.Grid", and so on.

method Control.Create ()

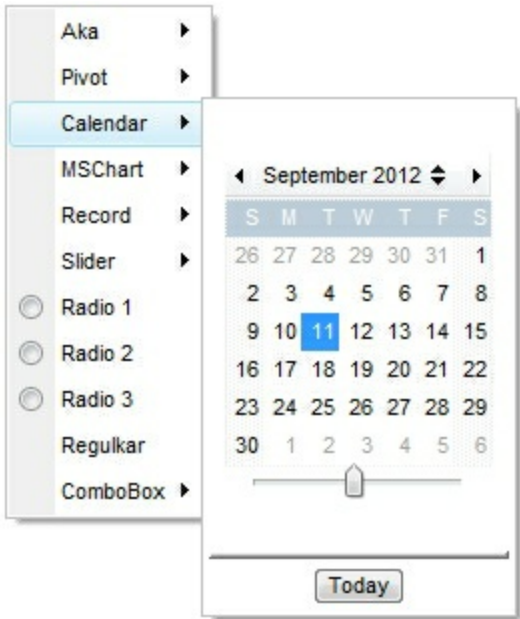
Creates the component.

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

The Create method creates the ActiveX control. The Create method creates the control based on its control's identifier. Use the [ControlID](#) and [LicenseKey](#) properties to specify the control's identifier and the runtime license key for the control, if required (please make sure that the runtime license key is not identical with your development license key). If the Create method fails, the [Object](#) property gets nothing. Use the Object property to access the ActiveX control's properties and methods. Use the [CloseOn](#) property to specify how the item that hosts an ActiveX control is closed using the mouse. Use the [Width](#) and [Height](#) properties to specify the size of the item that hosts the ActiveX control. The control fires the [OleEvent](#) event when an inside ActiveX control fires an event. 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 eXMenu control.

In case you are using the /NET assembly version, you can use the [Window](#) property to assign a Window/Control to an Item.

The following screen shot displays an item with an [ExCalendar](#) inside:



The following samples shows how to load an ActiveX control ([Exontrol.Calendar](#))

VBA (MS Access, Excell...)

```
With Ribbon1
    With .Items.Add("Calendar",3).SubControl
```

```
.Width = 256
.Height = 256
.ControlID = "Exontrol.Calendar"
.Create
End With
.Refresh
End With
```

VB6

```
With Ribbon1
  With .Items.Add("Calendar",3).SubControl
    .Width = 256
    .Height = 256
    .ControlID = "Exontrol.Calendar"
    .Create
  End With
  .Refresh
End With
```

VB.NET

```
With Exribbon1
  With .Items.Add("Calendar",3).SubControl
    .Width = 256
    .Height = 256
    .ControlID = "Exontrol.Calendar"
    .Create()
  End With
  .Refresh()
End With
```

VB.NET for /COM

```
With AxRibbon1
  With .Items.Add("Calendar",3).SubControl
    .Width = 256
    .Height = 256
```

```
.ControlID = "Exontrol.Calendar"  
.Create()  
End With  
.Refresh()  
End With
```

C++

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

```
    #import <ExRibbon.dll>  
    using namespace EXRIBBONLib;  
*/  
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-  
> GetControlUnknown();  
EXRIBBONLib::IControlPtr var_Control = spRibbon1->GetItems()-  
> Add(L"Calendar",long(3),vtMissing)-> GetSubControl();  
    var_Control->PutWidth(256);  
    var_Control->PutHeight(256);  
    var_Control->PutControlID(L"Exontrol.Calendar");  
    var_Control->Create();  
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IControlPtr var_Control = Ribbon1->Items-  
> Add(L"Calendar",TVariant(3),TNoParam())-> SubControl;  
    var_Control->Width = 256;  
    var_Control->Height = 256;  
    var_Control->ControlID = L"Exontrol.Calendar";  
    var_Control->Create();  
Ribbon1->Refresh();
```


C#

```
exontrol.EXRIBBONLib.Control var_Control =  
exribbon1.Items.Add("Calendar",3,null).SubControl;  
    var_Control.Width = 256;  
    var_Control.Height = 256;  
    var_Control.ControlID = "Exontrol.Calendar";  
    var_Control.Create();  
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    var var_Control = Ribbon1.Items.Add("Calendar",3,null).SubControl;  
        var_Control.Width = 256;  
        var_Control.Height = 256;  
        var_Control.ControlID = "Exontrol.Calendar";  
        var_Control.Create();  
        Ribbon1.Refresh();  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">
```

```

Function Init()
    With Ribbon1
        With .Items.Add("Calendar",3).SubControl
            .Width = 256
            .Height = 256
            .ControlID = "Exontrol.Calendar"
            .Create
        End With
    End With
    .Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Control var_Control =
axRibbon1.Items.Add("Calendar",3,null).SubControl;
var_Control.Width = 256;
var_Control.Height = 256;
var_Control.ControlID = "Exontrol.Calendar";
var_Control.Create();
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Control,com_Item;
    anytype var_Control,var_Item;
    ;

    super();

    var_Item =
    COM::createFromObject(exribbon1.Items()).Add("Calendar",COMVariant::createFromInt

```

```

com_Item = var_Item;
var_Control = com_Item.SubControl(); com_Control = var_Control;
com_Control.Width(256);
com_Control.Height(256);
com_Control.ControlID("Exontrol.Calendar");
com_Control.Create();
exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
with Items.Add('Calendar',TObject(3),Nil).SubControl do
begin
Width := 256;
Height := 256;
ControlID := 'Exontrol.Calendar';
Create();
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
with Items.Add('Calendar',OleVariant(3),Null).SubControl do
begin
Width := 256;
Height := 256;
ControlID := 'Exontrol.Calendar';
Create();
end;
Refresh();
end

```

VFP

```
with thisform.Ribbon1
  with .Items.Add("Calendar",3).SubControl
    .Width = 256
    .Height = 256
    .ControlID = "Exontrol.Calendar"
    .Create
  endwith
.Refresh
endwith
```

dBASE Plus

```
local oRibbon,var_Control

oRibbon = form.Activex1.nativeObject
var_Control = oRibbon.Items.Add("Calendar",3).SubControl
  var_Control.Width = 256
  var_Control.Height = 256
  var_Control.ControlID = "Exontrol.Calendar"
  var_Control.Create()
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
Dim var_Control as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Control = oRibbon.Items.Add("Calendar",3).SubControl
  var_Control.Width = 256
  var_Control.Height = 256
  var_Control.ControlID = "Exontrol.Calendar"
  var_Control.Create()
oRibbon.Refresh()
```

Visual Objects

local var_Control as IControl

```
var_Control := oDCOCX_Exontrol1:Items:Add("Calendar",3,nil):SubControl
var_Control.Width := 256
var_Control.Height := 256
var_Control.ControlID := "Exontrol.Calendar"
var_Control.Create()
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

OleObject oRibbon,var_Control

```
oRibbon = ole_1.Object
var_Control = oRibbon.Items.Add("Calendar",3).SubControl
var_Control.Width = 256
var_Control.Height = 256
var_Control.ControlID = "Exontrol.Calendar"
var_Control.Create()
oRibbon.Refresh()
```

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Variant voltems

Get ComItems to voltems

Handle holtems

Get Create (RefClass(cComItems)) to holtems

Set pvComObject of holtems to voltems

Variant voltem

Get ComAdd of holtems "Calendar" 3 Nothing to voltem

Handle holtem

Get Create (RefClass(cComItem)) to holtem

Set pvComObject of holtem to voltem

Variant voControl

```

Get ComSubControl of holtem to voControl
Handle hoControl
Get Create (RefClass(cComControl)) to hoControl
Set pvComObject of hoControl to voControl
    Set ComWidth of hoControl to 256
    Set ComHeight of hoControl to 256
    Set ComControlID of hoControl to "Exontrol.Calendar"
    Send ComCreate of hoControl
Send Destroy to hoControl
Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oControl
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oControl := oRibbon:Items():Add("Calendar",3):SubControl()
    oControl:Width := 256

```

oControl:Height := 256

oControl:ControlID := "Exontrol.Calendar"

oControl:Create()

oRibbon:Refresh()

oForm:Show()

DO WHILE nEvent != xbeP_Quit

nEvent := AppEvent(@mp1, @mp2, @oXbp)

oXbp:handleEvent(nEvent, mp1, mp2)

ENDDO

RETURN

property Control.Height as Long

Specifies the control's height.

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

By default, the Height property is 128 pixels. Use the Height property to specify the height of the inside control. The Height property has effect only if [Create](#) method is called after. Use the [Width](#) property to specify the control's width.

property Control.LicenseKey as String

Specifies the control's runtime license key.

Type	Description
String	A string expression that indicates the control's runtime license key.

The LicenseKey property must be set only if the control that you are going to use requires a runtime license key. Please contact the vendor of the control to know if the control requires a runtime license key. The control's runtime license key is not identical with your development license key. The LicenseKey property must be set before calling [Create](#) method. Please keep in mind that the vendor/provider of the ActiveX control you want to insert to an item is responsible for the control's runtime license key. Exontrol can provide the runtime license key for our components only.

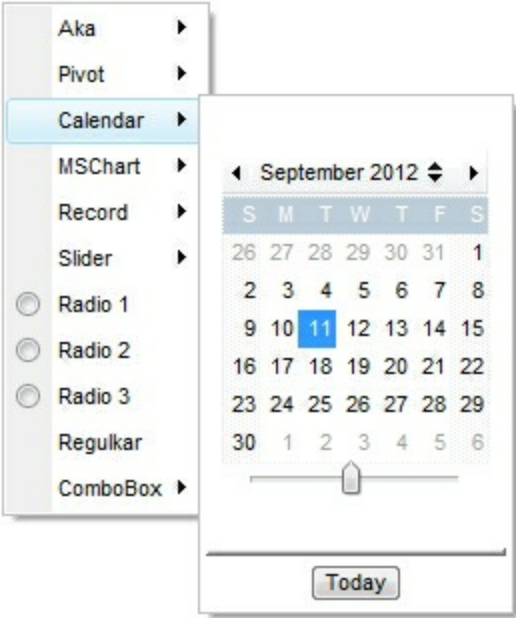
property Control.Object as Object

Gets the object.

Type	Description
Object	An Object created by the Create method.

Use the Object property to access to control's properties and methods. The type of the created object depends on [ControllID](#) property. The Object property gets nothing if no object was created. Use the Create method to create the inside ActiveX control. The control fires the [OleEvent](#) event when an inside ActiveX control fires an event. 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 eXMenu control.

The following screen shot displays an item with an [ExCalendar](#) inside:



property Control.Width as Long

Specifies the control's width.

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

By default, the Width property is 128 pixels. Use the Width property to specify the width of the inside control. The Width property has effect only if [Create](#) method is called after. Use the [Height](#) property to specify the control's height.

property Control.Window as Variant

Specifies the handle of the window to be hosted.

Type	Description
Variant	A Long expression that specifies the handle of the Window to be hosted by the Item.

Use the Window property to assign a Window to an item. The Window may be used by the /COM or /NET version by providing a valid handle to the window to be shown on the item. The /COM object may use the [Create](#) method to create an inside ActiveX control.

The following VB/.NET sample displays the form's PropertyGrid control to an Item (/NET version):

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles MyBase.Load

    ' Add 'exontrol.exribbon.dll' reference to your project.
    With Exribbon1
        With .Items
            With .Add("PropertiesGrid", 3).SubControl
                .Width = 256
                .Height = 312
                .Window = PropertyGrid1
            End With
        End With
    End With

    PropertyGrid1.SelectedObject = Exribbon1

End Sub
```

The following C# sample displays the form's PropertyGrid control to an Item (/NET version):

```
private void Form1_Load(object sender, EventArgs e)
{
    exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
    exontrol.EXRIBBONLib.Control var_Control = var_Items.Add("PropertiesGrid", 3,
```

```
null).SubControl;  
    var_Control.Width = 256;  
    var_Control.Height = 312;  
    var_Control.Window = propertyGrid1;  
  
    propertyGrid1.SelectedObject = exribbon1;  
  
}
```

Item object

The Item object holds information about an item in the ribbon control. The [Item](#) property searches recursively the item with giving identifier/caption. The Item object supports the following properties and methods:

Name	Description
Alignment	Retrieves or sets the item's caption alignment.
AllowEdit	Retrieves or sets a value indicating whether the item contains an edit control.
BackColor	Specifies the background color of the item.
Bold	Specifies whether the item's caption should appear in bold.
Caption	Retrieves or sets a value that indicates the item's caption.
CaptionWidth	Specifies the fixed width to display the item's caption.
Check	Retrieves or sets a value that indicates whether the item is of check type.
Checked	Retrieves or sets a value that indicates the item's state.
CloseOnClick	Specifies the way the owner menu is closed once the user clicks the item.
Cursor	Specifies the shape of the cursor when mouse hovers the item.
EditBorder	Specifies the border for the inside edit control.
EditCaption	Specifies the edit's caption when the item contains an edit control.
EditMask	Specifies the edit's mask when the item contains an masked edit control.
EditOption	Specifies different options for item's edit control.
EditValue	Specifies the edit's value when the item contains an edit control.
EditWidth	Specifies the width for the inside edit control.
Enabled	Retrieves or sets a value that indicates whether the item is enabled or disabled.
ForeColor	Specifies the foreground color of the item.
GroupPopup	Specifies whether the items of the sub-menu are grouped and displayed on the current item.

HotBackColor	Specifies the hot background color of the item (when the cursor hovers the item).
HTMLImage	Retrieves or sets a value that indicates the key of the image (HTMLPicture method) to be displayed on the item (left side).
ID	Retrieves or sets a value that specifies the item's identifier.
Image	Retrieves or sets a value that indicates the item's index image.
Italic	Specifies whether the item's caption should appear in italic.
ItemHeight	Specifies the fixed height to display the item.
Items	Retrieves an Items collection that indicates the item's sub menu. Retrieves Nothing, if the item contains no sub menu.
ItemType	Returns the type of the item.
Padding	Specifies the padding (space between the menu border and the item content) to display the item.
Parent	Gets the item's parent, if the current item belongs to a submenu/popup.
Position	Specifies the position of the item, within its collection.
Radio	Retrieves or sets a value that indicates whether the item is of radio type.
RadioGroup	Indicates the group of radio items that the current item belongs.
SelBackColor	Specifies the background color of the item when it is selected.
SelHotBackColor	Specifies the background color of the selected item when the cursor hovers it.
Shortcut	Specifies the key combination that the user can press to select the item quickly.
ShowAsButton	Specifies whether the item is shown as a button.
ShowAsDisabled	Specifies whether the item is shown as disabled.
ShowCheckedAsSelected	Specifies whether the checked item shows as selected.
ShowLocalPopup	Specifies whether the item's popup is shown as local. Clicking any item inside a local popup makes the popup itself to close including all its descendent sub-menus,

without closing any ascendant sub-menus.

[ShowPopupAlign](#)

Retrieves or sets a value that indicates how the item's sub-menu is aligned relative to the parent item.

[ShowPopupArrow](#)

Gets or sets a value that indicates whether an item that has a sub-menu shows or hides its popup arrow.

[ShowPopupAt](#)

Specifies the identifier of the item where the current item's submenu/popup is displayed.

[ShowPopupOffset](#)

Specifies the offset (horizontal,vertical) to display the item's submenu/popup relative to its default position.

[ShowPopupOnChecked](#)

Specifies whether the item's sub menu is shown only if the item is checked.

[Strikeout](#)

Specifies whether the item's caption should appear in strikeout.

[SubControl](#)

Retrieves the Control object that holds information about item's inside component.

[SubMenu](#)

Retrieves an Items collection that indicates the item's sub menu. Retrieves Nothing, if the item contains no sub menu.

[Tab](#)

Specifies the identifier of the item/tab where the current group popup is shown instead.

[Tooltip](#)

Specifies the item's tooltip.

[TooltipTitle](#)

Specifies the title of the item's tooltip.

[ToString](#)

Loads or saves the item using string representation.

[Underline](#)

Specifies whether the item's caption appears as underlined.

[Update](#)

Updates the item.

[UserData](#)

Associates an extra data to the object.

[Visible](#)

Specifies whether the item is visible or hidden.

property Item.Alignment as AlignmentEnum

Retrieves or sets the item's caption alignment.

Type	Description
AlignmentEnum	An AlignmentEnum expression that specifies the item's

The Alignment property specifies the item's alignment. The [Caption](#) property supports built-in HTML format, so you can use the <c> to centers the item's caption or <r> to align to the right the item's caption.

How do I align the item?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    .BackColor = RGB(250,250,250)
    .Add "Item"
    With .Add("Item")
      .Alignment = 1
      .CaptionWidth = 128
    End With
    With .Add("Item")
      .Alignment = 2
      .CaptionWidth = 128
    End With
    .Add("").ToString = "Item[align=1]"
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    .BackColor = RGB(250,250,250)
    .Add "Item"
    With .Add("Item")
      .Alignment = exCenter
    End With
  End With
End With
```

```

        .CaptionWidth = 128
    End With
    With .Add("Item")
        .Alignment = exRight
        .CaptionWidth = 128
    End With
    .Add("").ToString = "Item[align=1]"
End With
.Refresh
End With

```

VB.NET

```

With Exribbon1
    With .Items
        .BackColor = Color.FromArgb(250,250,250)
        .Add("Item")
        With .Add("Item")
            .Alignment = exontrol.EXRIBBONLib.AlignmentEnum.exCenter
            .CaptionWidth = 128
        End With
        With .Add("Item")
            .Alignment = exontrol.EXRIBBONLib.AlignmentEnum.exRight
            .CaptionWidth = 128
        End With
        .Add("").ToString = "Item[align=1]"
    End With
    .Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    With .Items
        .BackColor = RGB(250,250,250)
        .Add("Item")
        With .Add("Item")
            .Alignment = EXRIBBONLib.AlignmentEnum.exCenter

```

```

        .CaptionWidth = 128
    End With
    With .Add("Item")
        .Alignment = EXRIBBONLib.AlignmentEnum.exRight
        .CaptionWidth = 128
    End With
    .Add("").ToString = "Item[align=1]"
End With
.Refresh()
End With

```

C++

```

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
var_Items->PutBackColor(RGB(250,250,250));
var_Items->Add(L"Item",vtMissing,vtMissing);
EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Item",vtMissing,vtMissing);
var_Item->PutAlignment(EXRIBBONLib::exCenter);
var_Item->PutCaptionWidth(128);
EXRIBBONLib::IItemPtr var_Item1 = var_Items->Add(L"Item",vtMissing,vtMissing);
var_Item1->PutAlignment(EXRIBBONLib::exRight);
var_Item1->PutCaptionWidth(128);
var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"Item[align=1]");
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
var_Items->BackColor = RGB(250,250,250);
var_Items->Add(L"Item",TNoParam(),TNoParam());
Exribbonlib_tlb::IItemPtr var_Item = var_Items-
>Add(L"Item",TNoParam(),TNoParam());
var_Item->Alignment = Exribbonlib_tlb::AlignmentEnum::exCenter;
var_Item->CaptionWidth = 128;
Exribbonlib_tlb::IItemPtr var_Item1 = var_Items-
>Add(L"Item",TNoParam(),TNoParam());
var_Item1->Alignment = Exribbonlib_tlb::AlignmentEnum::exRight;
var_Item1->CaptionWidth = 128;
var_Items->Add(L"",TNoParam(),TNoParam())->ToString = L"Item[align=1]";
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
var_Items.BackColor = Color.FromArgb(250,250,250);
var_Items.Add("Item",null,null);
exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Item",null,null);
var_Item.Alignment = exontrol.EXRIBBONLib.AlignmentEnum.exCenter;
var_Item.CaptionWidth = 128;
exontrol.EXRIBBONLib.Item var_Item1 = var_Items.Add("Item",null,null);
var_Item1.Alignment = exontrol.EXRIBBONLib.AlignmentEnum.exRight;
var_Item1.CaptionWidth = 128;
var_Items.Add("",null,null).ToString = "Item[align=1]";
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()

```

```

{
var var_Items = Ribbon1.Items;
var var_Items.BackColor = 16448250;
var var_Items.Add("Item",null,null);
var var_Item = var_Items.Add("Item",null,null);
var var_Item.Alignment = 1;
var var_Item.CaptionWidth = 128;
var var_Item1 = var_Items.Add("Item",null,null);
var var_Item1.Alignment = 2;
var var_Item1.CaptionWidth = 128;
var var_Items.Add("",null,null).ToString = "Item[align=1]";
Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
With Ribbon1
With .Items
.BackColor = RGB(250,250,250)
.Add "Item"
With .Add("Item")
.Alignment = 1
.CaptionWidth = 128
End With
With .Add("Item")
.Alignment = 2
.CaptionWidth = 128
End With

```

```

        .Add("").ToString = "Item[align=1]"
    End With
    .Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.BackColor =
(uint)ColorTranslator.ToWin32(Color.FromArgb(250,250,250));
var_Items.Add("Item",null,null);
EXRIBBONLib.Item var_Item = var_Items.Add("Item",null,null);
var_Item.Alignment = EXRIBBONLib.AlignmentEnum.exCenter;
var_Item.CaptionWidth = 128;
EXRIBBONLib.Item var_Item1 = var_Items.Add("Item",null,null);
var_Item1.Alignment = EXRIBBONLib.AlignmentEnum.exRight;
var_Item1.CaptionWidth = 128;
var_Items.Add("",null,null).ToString = "Item[align=1]";
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Item1,com_Item2,com_Items;
    anytype var_Item,var_Item1,var_Item2,var_Items;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    com_Items.BackColor(WinApi::RGB2int(250,250,250));
    com_Items.Add("Item");
}

```

```

var_Item = com_Items.Add("Item"); com_Item = var_Item;
com_Item.Alignment(1/*exCenter*/);
com_Item.CaptionWidth(128);
var_Item1 = com_Items.Add("Item"); com_Item1 = var_Item1;
com_Item1.Alignment(2/*exRight*/);
com_Item1.CaptionWidth(128);
var_Item2 = COM::createFromObject(com_Items.Add("")); com_Item2 =
var_Item2;
com_Item2.ToString("Item[align=1]");
exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
  with Items do
  begin
    BackColor := $fafafa;
    Add('Item',Nil,Nil);
    with Add('Item',Nil,Nil) do
    begin
      Alignment := EXRIBBONLib.AlignmentEnum.exCenter;
      CaptionWidth := 128;
    end;
    with Add('Item',Nil,Nil) do
    begin
      Alignment := EXRIBBONLib.AlignmentEnum.exRight;
      CaptionWidth := 128;
    end;
    Add('',Nil,Nil).ToString := 'Item[align=1]';
  end;
  Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do

```

```

begin
  with Items do
    begin
      BackColor := $fafafa;
      Add('Item',Null,Null);
      with Add('Item',Null,Null) do
        begin
          Alignment := EXRIBBONLib_TLB.exCenter;
          CaptionWidth := 128;
        end;
      with Add('Item',Null,Null) do
        begin
          Alignment := EXRIBBONLib_TLB.exRight;
          CaptionWidth := 128;
        end;
      Add('',Null,Null).ToString := 'Item[align=1]';
    end;
    Refresh();
  end
end

```

VFP

```

with thisform.Ribbon1
  with .Items
    .BackColor = RGB(250,250,250)
    .Add("Item")
    with .Add("Item")
      .Alignment = 1
      .CaptionWidth = 128
    endwith
    with .Add("Item")
      .Alignment = 2
      .CaptionWidth = 128
    endwith
    .Add("").ToString = "Item[align=1]"
  endwith
.Refresh

```



```
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Item1,var_Item2,var_Items

oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
var_Items.BackColor = 0xfafafa
var_Items.Add("Item")
var_Item = var_Items.Add("Item")
var_Item.Alignment = 1
var_Item.CaptionWidth = 128
var_Item1 = var_Items.Add("Item")
var_Item1.Alignment = 2
var_Item1.CaptionWidth = 128
// var_Items.Add("").ToString = "Item[align=1]"
var_Item2 = var_Items.Add("")
with (oRibbon)
    TemplateDef = [Dim var_Item2]
    TemplateDef = var_Item2
    Template = [var_Item2.ToString = "Item[align=1]"]
endwith
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Item2 as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Items.BackColor = 16448250
var_Items.Add("Item")
```

```

var_Item = var_Items.Add("Item")
    var_Item.Alignment = 1
    var_Item.CaptionWidth = 128
var_Item1 = var_Items.Add("Item")
    var_Item1.Alignment = 2
    var_Item1.CaptionWidth = 128
' var_Items.Add("").ToString = "Item[align=1]"
var_Item2 = var_Items.Add("")
oRibbon.TemplateDef = "Dim var_Item2"
oRibbon.TemplateDef = var_Item2
oRibbon.Template = "var_Item2.ToString = \"Item[align=1]\""

oRibbon.Refresh()

```

Visual Objects

```

local var_Item,var_Item1 as Item
local var_Items as Items

var_Items := oDCOCX_Exontrol1:Items
var_Items.BackColor := RGB(250,250,250)
var_Items.Add("Item",nil,nil)
var_Item := var_Items.Add("Item",nil,nil)
    var_Item.Alignment := exCenter
    var_Item.CaptionWidth := 128
var_Item1 := var_Items.Add("Item",nil,nil)
    var_Item1.Alignment := exRight
    var_Item1.CaptionWidth := 128
var_Items.Add("",nil,nil):ToString := "Item[align=1]"
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```

OleObject oRibbon,var_Item,var_Item1,var_Items

oRibbon = ole_1.Object

```

```

var_Items = oRibbon.Items
var_Items.BackColor = RGB(250,250,250)
var_Items.Add("Item")
var_Item = var_Items.Add("Item")
    var_Item.Alignment = 1
    var_Item.CaptionWidth = 128
var_Item1 = var_Items.Add("Item")
    var_Item1.Alignment = 2
    var_Item1.CaptionWidth = 128
var_Items.Add("").ToString = "Item[align=1]"
oRibbon.Refresh()

```

Visual DataFlex

```

Procedure OnCreate
    Forward Send OnCreate
    Variant voltems
    Get ComItems to voltems
    Handle holtems
    Get Create (RefClass(cComItems)) to holtems
    Set pvComObject of holtems to voltems
        Set ComBackColor of holtems to (RGB(250,250,250))
        Get ComAdd of holtems "Item" Nothing Nothing to Nothing
        Variant voltem
        Get ComAdd of holtems "Item" Nothing Nothing to voltem
        Handle holtem
        Get Create (RefClass(cComItem)) to holtem
        Set pvComObject of holtem to voltem
            Set ComAlignment of holtem to OLEexCenter
            Set ComCaptionWidth of holtem to 128
        Send Destroy to holtem
        Variant voltem1
        Get ComAdd of holtems "Item" Nothing Nothing to voltem1
        Handle holtem1
        Get Create (RefClass(cComItem)) to holtem1
        Set pvComObject of holtem1 to voltem1

```

```

Set ComAlignment of holtem1 to OLEexRight
Set ComCaptionWidth of holtem1 to 128
Send Destroy to holtem1
Variant voltem2
Get ComAdd of holtems "" Nothing Nothing to voltem2
Handle holtem2
Get Create (RefClass(cComItem)) to holtem2
Set pvComObject of holtem2 to voltem2
Set ComToString of holtem2 to "Item[align=1]"
Send Destroy to holtem2
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem, oltem1
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oltems := oRibbon:Items()

```

```
    oltems:SetProperty("BackColor",AutomationTranslateColor(
GraMakeRGBColor ( { 250,250,250 } ) , .F. ))
    oltems:Add("Item")
    oltem := oltems:Add("Item")
    oltem:Alignment := 1/*exCenter*/
    oltem:CaptionWidth := 128
    oltem1 := oltems:Add("Item")
    oltem1:Alignment := 2/*exRight*/
    oltem1:CaptionWidth := 128
    oltems:Add(""):ToString := "Item[align=1]"
oRibbon:Refresh()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

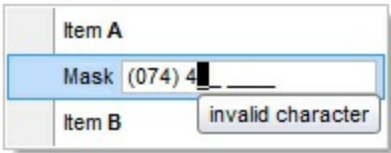
property Item.AllowEdit as AllowEditEnum

Retrieves or sets a value indicating whether the item contains an edit control.

Type	Description
AllowEditEnum	An AllowEditEnum expression that specifies whether the item displays an Edit field inside.

By default, the AllowEdit property is False, which indicates that the item displays no Edit field inside. Use the AllowEdit property to add a text-box inside the item, so the user can type any characters inside. The [EditCaption](#) property specifies the caption to be shown on the item's Edit text box. The [EditWidth](#) property specifies the width of the text-box inside the item. The [EditBorder](#) property specifies the border to be shown around the item's text box. You can use the [Get](#) method to collect all items of Edit type. The [EditChange](#) event notifies your application once the user alters the item's text-box caption. The [EditOption](#) property specifies different options to be used for a specified edit field. The [ShowAsButton](#) property specifies the whether the current item displays a button or a select button (drop down).

The following screen shot shows an item with a masking editor:



How can I add a vertical slider?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("Vertical")
      .ItemHeight = 128
      .AllowEdit = 1027 ' AllowEditEnum.exItemEditVertical Or
AllowEditEnum.exItemEditSlider
      .EditBorder = 0
      .EditWidth = 32
      .EditValue = 25
    End With
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    With .Add("Vertical")
      .ItemHeight = 128
      .AllowEdit = AllowEditEnum.exItemEditVertical Or
AllowEditEnum.exItemEditSlider
      .EditBorder = exEditBorderNone
      .EditWidth = 32
      .EditValue = 25
    End With
  End With
  .Refresh
End With
```

VB.NET

```
With Exribbon1
  With .Items
    With .Add("Vertical")
      .ItemHeight = 128
      .AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditVertical Or
exontrol.EXRIBBONLib.AllowEditEnum.exItemEditSlider
      .EditBorder = exontrol.EXRIBBONLib.EditBorderEnum.exEditBorderNone
      .EditWidth = 32
      .EditValue = 25
    End With
  End With
  .Refresh()
End With
```

VB.NET for /COM

```
With AxRibbon1
  With .Items
    With .Add("Vertical")
      .ItemHeight = 128
```

```

        .AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditVertical Or
EXRIBBONLib.AllowEditEnum.exItemEditSlider
        .EditBorder = EXRIBBONLib.EditBorderEnum.exEditBorderNone
        .EditWidth = 32
        .EditValue = 25
    End With
End With
.Refresh()
End With

```

C++

```

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Vertical",vtMissing,vtMissing);
        var_Item->PutItemHeight(128);
        var_Item-
> PutAllowEdit(EXRIBBONLib::AllowEditEnum(EXRIBBONLib::exItemEditVertical |
EXRIBBONLib::exItemEditSlider));
        var_Item->PutEditBorder(EXRIBBONLib::exEditBorderNone);
        var_Item->PutEditWidth(32);
        var_Item->PutEditValue(long(25));
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    Exribbonlib_tlb::IItemPtr var_Item = var_Items-

```



```

>Add(L"Vertical",TNoParam(),TNoParam());
    var_Item->ItemHeight = 128;
    var_Item->AllowEdit = Exribbonlib_tlb::AllowEditEnum::exItemEditVertical |
Exribbonlib_tlb::AllowEditEnum::exItemEditSlider;
    var_Item->EditBorder = Exribbonlib_tlb::EditBorderEnum::exEditBorderNone;
    var_Item->EditWidth = 32;
    var_Item->set_EditValue(TVariant(25));
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
    exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Vertical",null,null);
        var_Item.ItemHeight = 128;
        var_Item.AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditVertical |
exontrol.EXRIBBONLib.AllowEditEnum.exItemEditSlider;
        var_Item.EditBorder = exontrol.EXRIBBONLib.EditBorderEnum.exEditBorderNone;
        var_Item.EditWidth = 32;
        var_Item.EditValue = 25;
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
        var var_Item = var_Items.Add("Vertical",null,null);
            var_Item.ItemHeight = 128;
            var_Item.AllowEdit = 1027;
            var_Item.EditBorder = 0;
            var_Item.EditWidth = 32;

```

```

        var_Item.EditValue = 25;
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            With .Add("Vertical")
                .ItemHeight = 128
                .AllowEdit = 1027 ' AllowEditEnum.exItemEditVertical Or
AllowEditEnum.exItemEditSlider
                .EditBorder = 0
                .EditWidth = 32
                .EditValue = 25
            End With
        End With
    End With
    .Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
EXRIBBONLib.Item var_Item = var_Items.Add("Vertical",null,null);
var_Item.ItemHeight = 128;

```

```

var_Item.AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditVertical |
EXRIBBONLib.AllowEditEnum.exItemEditSlider;
var_Item.EditBorder = EXRIBBONLib.EditBorderEnum.exEditBorderNone;
var_Item.EditWidth = 32;
var_Item.EditValue = 25;
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("Vertical"); com_Item = var_Item;
    com_Item.ItemHeight(128);
    com_Item.AllowEdit(1027/*exItemEditVertical | exItemEditSlider*/);
    com_Item.EditBorder(0/*exEditBorderNone*/);
    com_Item.EditWidth(32);
    com_Item.EditValue(COMVariant::createFromInt(25));
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        with Add('Vertical',Nil,Nil) do
        begin
            ItemHeight := 128;
            AllowEdit := Integer(EXRIBBONLib.AllowEditEnum.exItemEditVertical) Or

```

```

Integer(EXRIBBONLib.AllowEditEnum.exItemEditSlider);
    EditBorder := EXRIBBONLib.EditBorderEnum.exEditBorderNone;
    EditWidth := 32;
    EditValue := TObject(25);
end;
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin
        with Add('Vertical',Null,Null) do
        begin
            ItemHeight := 128;
            AllowEdit := Integer(EXRIBBONLib_TLB.exItemEditVertical) Or
Integer(EXRIBBONLib_TLB.exItemEditSlider);
            EditBorder := EXRIBBONLib_TLB.exEditBorderNone;
            EditWidth := 32;
            EditValue := OleVariant(25);
        end;
        end;
        Refresh();
    end
end

```

VFP

```

with thisform.Ribbon1
    with .Items
        with .Add("Vertical")
            .ItemHeight = 128
            .AllowEdit = 1027 && AllowEditEnum.exItemEditVertical Or
AllowEditEnum.exItemEditSlider
            .EditBorder = 0
            .EditWidth = 32

```

```
.EditValue = 25  
endwith  
endwith  
.Refresh  
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Items  
  
oRibbon = form.ActiveX1.nativeObject  
var_Items = oRibbon.Items  
var_Item = var_Items.Add("Vertical")  
var_Item.ItemHeight = 128  
var_Item.AllowEdit = 1027 /*exItemEditVertical | exItemEditSlider*/  
var_Item.EditBorder = 0  
var_Item.EditWidth = 32  
var_Item.EditValue = 25  
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P  
Dim var_Item as P  
Dim var_Items as P  
  
oRibbon = topparent:CONTROL_ACTIVEX1.activex  
var_Items = oRibbon.Items  
var_Item = var_Items.Add("Vertical")  
var_Item.ItemHeight = 128  
var_Item.AllowEdit = 1027 'exItemEditVertical + exItemEditSlider  
var_Item.EditBorder = 0  
var_Item.EditWidth = 32  
var_Item.EditValue = 25  
oRibbon.Refresh()
```

Visual Objects

```
local var_Item as Item
local var_Items as Items

var_Items := oDCOCX_Exontrol1:Items
  var_Item := var_Items:Add("Vertical",nil,nil)
    var_Item:ItemHeight := 128
    var_Item:AllowEdit := exItemEditVertical | exItemEditSlider
    var_Item:EditBorder := exEditBorderNone
    var_Item:EditWidth := 32
    var_Item:EditValue := 25
oDCOCX_Exontrol1:Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Items

oRibbon = ole_1.Object
var_Items = oRibbon.Items
  var_Item = var_Items.Add("Vertical")
    var_Item.ItemHeight = 128
    var_Item.AllowEdit = 1027 /*exItemEditVertical | exItemEditSlider*/
    var_Item.EditBorder = 0
    var_Item.EditWidth = 32
    var_Item.EditValue = 25
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
  Forward Send OnCreate
  Variant voltems
  Get ComItems to voltems
  Handle holtems
  Get Create (RefClass(cComItems)) to holtems
```

```

Set pvComObject of holtems to voltems
Variant voltem
Get ComAdd of holtems "Vertical" Nothing Nothing to voltem
Handle holtem
Get Create (RefClass(cComltem)) to holtem
Set pvComObject of holtem to voltem
    Set ComltemHeight of holtem to 128
    Set ComAllowEdit of holtem to (OLEexltemEditVertical +
OLEexltemEditSlider)
    Set ComEditBorder of holtem to OLEexEditBorderNone
    Set ComEditWidth of holtem to 32
    Set ComEditValue of holtem to 25
    Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( „{100,100}, {640,480}„, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/

```

```
oRibbon:create(,, {10,60},{610,370} )
```

```
oItems := oRibbon:Items()
```

```
oItem := oItems:Add("Vertical")
```

```
oItem:ItemHeight := 128
```

```
oItem:AllowEdit := 1027/*exItemEditVertical+exItemEditSlider*/
```

```
oItem:EditBorder := 0/*exEditBorderNone*/
```

```
oItem:EditWidth := 32
```

```
oItem:EditValue := 25
```

```
oRibbon:Refresh()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```


property Item.BackColor as Color

Specifies the item's background color of the item.

Type	Description
Color	A Color expression that specifies the item's 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.

The BackColor property specifies a different background color or a visual appearance for the item. The [Caption](#) property indicates the item's caption to be shown on the item. You can use the <bgcolor> HTML tag in the Caption property to specify a different background color for parts of the caption. The [ForeColor](#) property specifies the item's foreground color. The [SelBackColor](#) property specifies the item's background color when it is selected or highlighted. The [HotBackColor](#) property specifies a different background color or a visual appearance for the item, when the cursor hovers it. The [SelHotBackColor](#) property specifies a different background color or a visual appearance for the item, when item is selected / checked, and the cursor hovers it. The [SelBackColor](#) property specifies a different background color or a visual appearance for the item, when item is selected / checked.

How can I change the background color for a specified item?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    .Add "Item 1"
    .Add("Item 2").BackColor = RGB(255,0,0)
    .Add "Item 3"
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
```

```
.Add "Item 1"  
.Add("Item 2").BackColor = RGB(255,0,0)  
.Add "Item 3"  
End With  
.Refresh  
End With
```

VB.NET

```
With Exribbon1  
    With .Items  
        .Add("Item 1")  
        .Add("Item 2").BackColor = Color.FromArgb(255,0,0)  
        .Add("Item 3")  
    End With  
    .Refresh()  
End With
```

VB.NET for /COM

```
With AxRibbon1  
    With .Items  
        .Add("Item 1")  
        .Add("Item 2").BackColor = RGB(255,0,0)  
        .Add("Item 3")  
    End With  
    .Refresh()  
End With
```

C++

```
/*  
Copy and paste the following directives to your header file as  
it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control  
Library'  
  
#import <ExRibbon.dll>  
using namespace EXRIBBONLib;
```

```

*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    var_Items->Add(L"Item 1",vtMissing,vtMissing);
    var_Items->Add(L"Item 2",vtMissing,vtMissing)->PutBackColor(
RGB(255,0,0));
    var_Items->Add(L"Item 3",vtMissing,vtMissing);
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    var_Items->Add(L"Item 1",TNoParam(),TNoParam());
    var_Items->Add(L"Item 2",TNoParam(),TNoParam())->BackColor =
RGB(255,0,0);
    var_Items->Add(L"Item 3",TNoParam(),TNoParam());
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
    var_Items.Add("Item 1",null,null);
    var_Items.Add("Item 2",null,null).BackColor = Color.FromArgb(
255,0,0);
    var_Items.Add("Item 3",null,null);
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;

```

```

var_Items.Add("Item 1",null,null);
var_Items.Add("Item 2",null,null).BackColor = 255;
var_Items.Add("Item 3",null,null);
Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            .Add "Item 1"
            .Add("Item 2").BackColor = RGB(255,0,0)
            .Add "Item 3"
        End With
        .Refresh
    End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.Add("Item 1",null,null);
var_Items.Add("Item 2",null,null).BackColor =
(uint)ColorTranslator.ToWin32(Color.FromArgb(255,0,0));
var_Items.Add("Item 3",null,null);
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    com_Items.Add("Item 1");
    var_Item = COM::createFromObject(com_Items.Add("Item 2")); com_Item =
var_Item;
    com_Item.BackColor(WinApi::RGB2int(255,0,0));
    com_Items.Add("Item 3");
    exribbon1.Refresh();
}
```

Delphi 8 (.NET only)

```
with AxRibbon1 do
begin
    with Items do
    begin
        Add('Item 1',Nil,Nil);
        Add('Item 2',Nil,Nil).BackColor := $ff;
        Add('Item 3',Nil,Nil);
    end;
    Refresh();
end
```

Delphi (standard)

```
with Ribbon1 do
begin
```

```
with Items do
begin
  Add('Item 1',Null,Null);
  Add('Item 2',Null,Null).BackColor := $ff;
  Add('Item 3',Null,Null);
end;
Refresh();
end
```

VFP

```
with thisform.Ribbon1
with .Items
  .Add("Item 1")
  .Add("Item 2").BackColor = RGB(255,0,0)
  .Add("Item 3")
endwith
.Refresh
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Items

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
var_Items.Add("Item 1")
// var_Items.Add("Item 2").BackColor = 0xff
var_Item = var_Items.Add("Item 2")
with (oRibbon)
  TemplateDef = [Dim var_Item]
  TemplateDef = var_Item
  Template = [var_Item.BackColor = 0xff]
endwith
var_Items.Add("Item 3")
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
Dim var_Item as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
  var_Items.Add("Item 1")
  ' var_Items.Add("Item 2").BackColor = 255
var_Item = var_Items.Add("Item 2")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.BackColor = 255"

  var_Items.Add("Item 3")
oRibbon.Refresh()
```

Visual Objects

```
local var_Items as IItems

var_Items := oDCOCX_Exontrol1.Items
  var_Items.Add("Item 1",nil,nil)
  var_Items.Add("Item 2",nil,nil):BackColor := RGB(255,0,0)
  var_Items.Add("Item 3",nil,nil)
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Items

oRibbon = ole_1.Object
var_Items = oRibbon.Items
  var_Items.Add("Item 1")
  var_Items.Add("Item 2").BackColor = RGB(255,0,0)
```

```
var_Items.Add("Item 3")
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
  Forward Send OnCreate
  Variant voltems
  Get ComItems to voltems
  Handle holtems
  Get Create (RefClass(cComItems)) to holtems
  Set pvComObject of holtems to voltems
    Get ComAdd of holtems "Item 1" Nothing Nothing to Nothing
    Variant voltem
    Get ComAdd of holtems "Item 2" Nothing Nothing to voltem
    Handle holtem
    Get Create (RefClass(cComItem)) to holtem
    Set pvComObject of holtem to voltem
      Set ComBackColor of holtem to (RGB(255,0,0))
    Send Destroy to holtem
    Get ComAdd of holtems "Item 3" Nothing Nothing to Nothing
  Send Destroy to holtems
  Send ComRefresh
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oltems
  LOCAL oRibbon

  oForm := XbpDialog():new( AppDesktop() )
```



```

oForm:drawingArea:clipChildren := .T.
oForm:create( ,, {100,100}, {640,480} ,, .F. )
oForm:close := {|| PostAppEvent( xbeP_Quit )}

oRibbon := XbpActiveXControl():new( oForm:drawingArea )
oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
oRibbon:create(,, {10,60},{610,370} )

    oltems := oRibbon:Items()
    oltems:Add("Item 1")
    oltems:Add("Item 2"):SetProperty("BackColor",AutomationTranslateColor(
GraMakeRGBColor ( { 255,0,0 } ) , .F. ))
    oltems:Add("Item 3")
    oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

property Item.Bold as Boolean

Specifies whether the item's caption should appear in bold.

Type	Description
Boolean	A Boolean expression that specifies whether the item's caption is shown in bold.

By default, the Bold property is False. Use the Bold property to show the item's caption in bold. The [Caption](#) property indicates the HTML caption to be shown on the item. The HTML tag can be used on the item's Caption property to specify different parts of the caption to be shown in bold.

How can I show the item in bold?

VBA (MS Access, Excell...)

```
With Ribbon1
    With .Items
        .Add("Item").Bold = True
        .Add "<b>Item</b>"
        .Add("").ToString = "Item[bld]"
    End With
    .Refresh
End With
```

VB6

```
With Ribbon1
    With .Items
        .Add("Item").Bold = True
        .Add "<b>Item</b>"
        .Add("").ToString = "Item[bld]"
    End With
    .Refresh
End With
```

VB.NET

```
With Exribbon1
    With .Items
```

```

.Add("Item").Bold = True
.Add("<b>Item</b>")
.Add("").ToString = "Item[bld]"
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
  With .Items
    .Add("Item").Bold = True
    .Add("<b>Item</b>")
    .Add("").ToString = "Item[bld]"
  End With
  .Refresh()
End With

```

C++

```

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

  #import <ExRibbon.dll>
  using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
var_Items->Add(L"Item",vtMissing,vtMissing)->PutBold(VARIANT_TRUE);
var_Items->Add(L"<b>Item</b>",vtMissing,vtMissing);
var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"Item[bld]");
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
var_Items->Add(L"Item",TNoParam(),TNoParam())->Bold = true;
var_Items->Add(L"<b>Item</b>",TNoParam(),TNoParam());
var_Items->Add(L"",TNoParam(),TNoParam())->ToString = L"Item[bld]";
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
var_Items.Add("Item",null,null).Bold = true;
var_Items.Add("<b>Item</b>",null,null);
var_Items.Add("",null,null).ToString = "Item[bld]";
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var_Items.Add("Item",null,null).Bold = true;
    var_Items.Add("<b>Item</b>",null,null);
    var_Items.Add("",null,null).ToString = "Item[bld]";
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>

```

```
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"></OBJECT>
```

```
<SCRIPT LANGUAGE="VBScript">
```

```
Function Init()
```

```
    With Ribbon1
```

```
        With .Items
```

```
            .Add("Item").Bold = True
```

```
            .Add "<b>Item</b>"
```

```
            .Add("").ToString = "Item[bld]"
```

```
        End With
```

```
        .Refresh
```

```
    End With
```

```
End Function
```

```
</SCRIPT>
```

```
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.Add("Item",null,null).Bold = true;
var_Items.Add("<b>Item</b>",null,null);
var_Items.Add("",null,null).ToString = "Item[bld]";
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
```

```

    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
    com_Item.Bold(true);
    com_Items.Add("<b>Item</b>");
    var_Item = COM::createFromObject(com_Items.Add("")); com_Item = var_Item;
    com_Item.ToString("Item[bld]");
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
  with Items do
  begin
    Add('Item',Nil,Nil).Bold := True;
    Add('<b>Item</b>',Nil,Nil);
    Add('',Nil,Nil).ToString := 'Item[bld]';
  end;
  Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  with Items do
  begin
    Add('Item',Null,Null).Bold := True;
    Add('<b>Item</b>',Null,Null);
    Add('',Null,Null).ToString := 'Item[bld]';
  end;
  Refresh();
end

```

VFP

```

with thisform.Ribbon1

```

```

with .Items
    .Add("Item").Bold = .T.
    .Add("<b>Item</b>")
    .Add("").ToString = "Item[bld]"
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Items

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
// var_Items.Add("Item").Bold = true
var_Item = var_Items.Add("Item")
with (oRibbon)
    TemplateDef = [Dim var_Item]
    TemplateDef = var_Item
    Template = [var_Item.Bold = true]
endwith
var_Items.Add("<b>Item</b>")
// var_Items.Add("").ToString = "Item[bld]"
var_Item1 = var_Items.Add("")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.ToString = "Item[bld]"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P

```

```

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
' var_Items.Add("Item").Bold = .t.
var_Item = var_Items.Add("Item")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.Bold = True"

var_Items.Add("<b>Item</b>")
' var_Items.Add("").ToString = "Item[bld]"
var_Item1 = var_Items.Add("")
oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1
oRibbon.Template = "var_Item1.ToString = \"Item[bld]\""

oRibbon.Refresh()

```

Visual Objects

```

local var_Items as IItems

var_Items := oDCOCX_Exontrol1.Items
var_Items.Add("Item",nil,nil):Bold := true
var_Items.Add("<b>Item</b>",nil,nil)
var_Items.Add("",nil,nil):ToString := "Item[bld]"
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```

OleObject oRibbon,var_Items

oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Items.Add("Item").Bold = true
var_Items.Add("<b>Item</b>")

```



```
var_Items.Add("").ToString = "Item[bld]"  
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate  
  Forward Send OnCreate  
  Variant voltems  
  Get ComItems to voltems  
  Handle holtems  
  Get Create (RefClass(cComItems)) to holtems  
  Set pvComObject of holtems to voltems  
    Variant voltem  
    Get ComAdd of holtems "Item" Nothing Nothing to voltem  
    Handle holtem  
    Get Create (RefClass(cComItem)) to holtem  
    Set pvComObject of holtem to voltem  
      Set ComBold of holtem to True  
    Send Destroy to holtem  
    Get ComAdd of holtems "<b>Item</b>" Nothing Nothing to Nothing  
    Variant voltem1  
    Get ComAdd of holtems "" Nothing Nothing to voltem1  
    Handle holtem1  
    Get Create (RefClass(cComItem)) to holtem1  
    Set pvComObject of holtem1 to voltem1  
      Set ComToString of holtem1 to "Item[bld]"  
    Send Destroy to holtem1  
  Send Destroy to holtems  
  Send ComRefresh  
End_Procedure
```

XBase++

```
#include "AppEvent.ch"  
#include "ActiveX.ch"  
  
PROCEDURE Main
```

LOCAL oForm

LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

LOCAL oltems

LOCAL oRibbon

oForm := XbpDialog():new(AppDesktop())

oForm:drawingArea:clipChildren := .T.

oForm:create(„{100,100}, {640,480}„, .F.)

oForm:close := {|| PostAppEvent(xbeP_Quit)}

oRibbon := XbpActiveXControl():new(oForm:drawingArea)

oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-CFBE431702E2}*/

oRibbon:create(„ {10,60},{610,370})

oltems := oRibbon:Items()

oltems:Add("Item"):Bold := .T.

oltems:Add("Item")

oltems:Add(""):ToString := "Item[bld]"

oRibbon:Refresh()

oForm:Show()

DO WHILE nEvent != xbeP_Quit

nEvent := AppEvent(@mp1, @mp2, @oXbp)

oXbp:handleEvent(nEvent, mp1, mp2)

ENDDO

RETURN

property Item.Caption as String

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

Type	Description
String	A String expression that specifies the HTML caption to be displayed on the item.

Use the Caption property to specify the item's caption. Use the [UserData](#) property to associate any extra data to your items. Use the [Tooltip](#) property to specify the item's tooltip which can be shown when the cursor hovers the item. Use the [Check](#) property to assign a check-box to the item. Use the [Radio](#) property to assign a radio-button to the item. The [ForeColor](#) property of the Item object specifies a different foreground color for the entire item. The [BackColor](#) property of the Item object specifies a different background color / visual appearance for the entire item. The [Item](#) property searches recursively the item with giving identifier/caption. The [AllowEdit](#) property assigns an editor to an item. The [CaptionWidth](#) property specifies the fixed width to display the item's caption.

The Caption property supports the following HTML tags:

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

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

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

color from the current text color to gray (808080). For instance the "<gradient-center" 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 Item.CaptionWidth as Long

Specifies the fixed width to display the item's caption.

Type	Description
Long	A Long expression that specifies the width to display the item's caption.

By default, the CaptionWidth property is -1. If the CaptionWidth is negative, the caption expands the size of the item to fit entirely. If the CaptionWidth property is positive, it indicates the width to display the item's caption. The [Caption](#) property specifies the HTML caption to be displayed on the item. For instance, you can use the CaptionWidth to align editors of the items. The [ItemHeight](#) property specifies the height to show the item.

How can I specify the width of the item?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    .BackColor = RGB(250,250,250)
    .Add("Item").CaptionWidth = 128
    .Add("").ToString = "Item[captionwidth=128]"
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    .BackColor = RGB(250,250,250)
    .Add("Item").CaptionWidth = 128
    .Add("").ToString = "Item[captionwidth=128]"
  End With
  .Refresh
End With
```

VB.NET

```
With Exribbon1
```

```

With .Items
    .BackColor = Color.FromArgb(250,250,250)
    .Add("Item").CaptionWidth = 128
    .Add("").ToString = "Item[captionwidth=128]"
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    With .Items
        .BackColor = RGB(250,250,250)
        .Add("Item").CaptionWidth = 128
        .Add("").ToString = "Item[captionwidth=128]"
    End With
    .Refresh()
End With

```

C++

```

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
> GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
var_Items->PutBackColor(RGB(250,250,250));
var_Items->Add(L"Item",vtMissing,vtMissing)->PutCaptionWidth(128);
var_Items->Add(L"",vtMissing,vtMissing)-
> PutToString(L"Item[captionwidth=128]");
spRibbon1->Refresh();

```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
var_Items->BackColor = RGB(250,250,250);
var_Items->Add(L"Item",TNoParam(),TNoParam())->CaptionWidth = 128;
var_Items->Add(L"",TNoParam(),TNoParam())->ToString =
L"Item[captionwidth=128]";
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
var_Items.BackColor = Color.FromArgb(250,250,250);
var_Items.Add("Item",null,null).CaptionWidth = 128;
var_Items.Add("",null,null).ToString = "Item[captionwidth=128]";
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var_Items.BackColor = 16448250;
    var_Items.Add("Item",null,null).CaptionWidth = 128;
    var_Items.Add("",null,null).ToString = "Item[captionwidth=128]";
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>
```

VBScript


```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            .BackColor = RGB(250,250,250)
            .Add("Item").CaptionWidth = 128
            .Add("").ToString = "Item[captionwidth=128]"
        End With
        .Refresh
    End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.BackColor =
(uint)ColorTranslator.ToWin32(Color.FromArgb(250,250,250));
var_Items.Add("Item",null,null).CaptionWidth = 128;
var_Items.Add("",null,null).ToString = "Item[captionwidth=128]";
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

```

```

super();

var_Items = exribbon1.Items(); com_Items = var_Items;
    com_Items.BackColor(WinApi::RGB2int(250,250,250));
    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
    com_Item.CaptionWidth(128);
    var_Item = COM::createFromObject(com_Items.Add("")); com_Item = var_Item;
    com_Item.ToString("Item[captionwidth=128]");
exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        BackColor := $fafafa;
        Add('Item',Nil,Nil).CaptionWidth := 128;
        Add('',Nil,Nil).ToString := 'Item[captionwidth=128]';
    end;
    Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin
        BackColor := $fafafa;
        Add('Item',Null,Null).CaptionWidth := 128;
        Add('',Null,Null).ToString := 'Item[captionwidth=128]';
    end;
    Refresh();
end

```

VFP

```
with thisform.Ribbon1
  with .Items
    .BackColor = RGB(250,250,250)
    .Add("Item").CaptionWidth = 128
    .Add("").ToString = "Item[captionwidth=128]"
  endwhile
  .Refresh
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Item1,var_Items

oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
var_Items.BackColor = 0xfafafa
// var_Items.Add("Item").CaptionWidth = 128
var_Item = var_Items.Add("Item")
with (oRibbon)
  TemplateDef = [Dim var_Item]
  TemplateDef = var_Item
  Template = [var_Item.CaptionWidth = 128]
endwith
// var_Items.Add("").ToString = "Item[captionwidth=128]"
var_Item1 = var_Items.Add("")
with (oRibbon)
  TemplateDef = [Dim var_Item1]
  TemplateDef = var_Item1
  Template = [var_Item1.ToString = "Item[captionwidth=128]"]
endwith
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
```

```
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P
```

```
oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Items.BackColor = 16448250
' var_Items.Add("Item").CaptionWidth = 128
var_Item = var_Items.Add("Item")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.CaptionWidth = 128"

' var_Items.Add("").ToString = "Item[captionwidth=128]"
var_Item1 = var_Items.Add("")
oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1
oRibbon.Template = "var_Item1.ToString = \"Item[captionwidth=128]\""

oRibbon.Refresh()
```

Visual Objects

```
local var_Items as IItems

var_Items := oDCOCX_Exontrol1.Items
var_Items.BackColor := RGB(250,250,250)
var_Items.Add("Item",nil,nil):CaptionWidth := 128
var_Items.Add("",nil,nil):ToString := "Item[captionwidth=128]"
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Items

oRibbon = ole_1.Object
```

```
var_Items = oRibbon.Items
var_Items.BackColor = RGB(250,250,250)
var_Items.Add("Item").CaptionWidth = 128
var_Items.Add("").ToString = "Item[captionwidth=128]"
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
  Forward Send OnCreate
  Variant voltems
  Get ComItems to voltems
  Handle holtems
  Get Create (RefClass(cComItems)) to holtems
  Set pvComObject of holtems to voltems
    Set ComBackColor of holtems to (RGB(250,250,250))
  Variant voltem
  Get ComAdd of holtems "Item" Nothing Nothing to voltem
  Handle holtem
  Get Create (RefClass(cComItem)) to holtem
  Set pvComObject of holtem to voltem
    Set ComCaptionWidth of holtem to 128
  Send Destroy to holtem
  Variant voltem1
  Get ComAdd of holtems "" Nothing Nothing to voltem1
  Handle holtem1
  Get Create (RefClass(cComItem)) to holtem1
  Set pvComObject of holtem1 to voltem1
    Set ComToString of holtem1 to "Item[captionwidth=128]"
  Send Destroy to holtem1
  Send Destroy to holtems
  Send ComRefresh
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
    LOCAL oForm
```

```
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
    LOCAL oltems
```

```
    LOCAL oRibbon
```

```
    oForm := XbpDialog():new( AppDesktop() )
```

```
    oForm:drawingArea:clipChildren := .T.
```

```
    oForm:create( ,, {100,100}, {640,480},,, .F. )
```

```
    oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
```

```
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-  
CFBE431702E2}*/
```

```
    oRibbon:create(,, {10,60},{610,370} )
```

```
        oltems := oRibbon:Items()
```

```
            oltems:SetProperty("BackColor",AutomationTranslateColor(  
GraMakeRGBColor ( { 250,250,250 } ) , .F. ))
```

```
            oltems:Add("Item"):CaptionWidth := 128
```

```
            oltems:Add(""):ToString := "Item[captionwidth=128]"
```

```
        oRibbon:Refresh()
```

```
    oForm:Show()
```

```
    DO WHILE nEvent != xbeP_Quit
```

```
        nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
        oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
    ENDDO
```

```
RETURN
```

property Item.Check as Boolean

Retrieves or sets a value that indicates whether the item is of check type.

Type	Description
Boolean	A Boolean expression that specifies whether the item displays a check box.

The Check property indicates whether the current item displays a check box. The [Checked](#) property specifies whether the item is checked or un-checked. The [Radio](#) property specifies whether the item displays a radio-button. The [RadioGroup](#) property specifies a group of radio-buttons. A radio group allows a single radio-item to be checked inside. The [GetChecked](#) property gets a collection of checked items. The [GetUnchecked](#) property gets a collection of unchecked items. The [GetRadio](#) method gets a safe array with the radio-items being checked within a radio group. Use the [Background\(exCheckBoxState0\)/Background\(exCheckBoxState1\)](#) property to specify the visual appearance of the check-boxes in the control. Use the [UseVisualTheme](#) property to specify whether the visual appearance for the check-boxes to be as indicated by the current XP theme. Use the [ShowCheckedAsSelected](#) property on True, to show the checked items as selected.

How can I add check-buttons?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("",2)
      .GroupPopup = 3 ' GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
    With .Items
      With .Add("Check 1")
        .Check = True
        .Checked = True
      End With
      .Add("Check 2").Check = True
    End With
  End With
End With
.Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    With .Add("",2)
      .GroupPopup = GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
    With .Items
      With .Add("Check 1")
        .Check = True
        .Checked = True
      End With
      .Add("Check 2").Check = True
    End With
  End With
End With
.Refresh
End With
```

VB.NET

```
With Exribbon1
  With .Items
    With .Add("",2)
      .GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup
    With .Items
      With .Add("Check 1")
        .Check = True
        .Checked = True
      End With
      .Add("Check 2").Check = True
    End With
  End With
End With
.Refresh()
End With
```


VB.NET for /COM

```
With AxRibbon1
  With .Items
    With .Add("",2)
      .GroupPopup = EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or
EXRIBBONLib.GroupPopupEnum.exGroupPopup
    With .Items
      With .Add("Check 1")
        .Check = True
        .Checked = True
      End With
      .Add("Check 2").Check = True
    End With
  End With
End With
.Refresh()
End With
```

C++

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

  #import <ExRibbon.dll>
  using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
  EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"",long(2),vtMissing);
  var_Item-
>PutGroupPopup(EXRIBBONLib::GroupPopupEnum(EXRIBBONLib::exNoGroupPopupFrame
| EXRIBBONLib::exGroupPopup));
  EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();
  EXRIBBONLib::IItemPtr var_Item1 = var_Items1->Add(L"Check
```

```

1",vtMissing,vtMissing);
    var_Item1->PutCheck(VARIANT_TRUE);
    var_Item1->PutChecked(VARIANT_TRUE);
    var_Items1->Add(L"Check 2",vtMissing,vtMissing)-
> PutCheck(VARIANT_TRUE);
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    Exribbonlib_tlb::IItemPtr var_Item = var_Items->Add(L"",TVariant(2),TNoParam());
    var_Item->GroupPopup =
Exribbonlib_tlb::GroupPopupEnum::exNoGroupPopupFrame |
Exribbonlib_tlb::GroupPopupEnum::exGroupPopup;
    Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;
    Exribbonlib_tlb::IItemPtr var_Item1 = var_Items1->Add(L"Check
1",TNoParam(),TNoParam());
    var_Item1->Check = true;
    var_Item1->Checked = true;
    var_Items1->Add(L"Check 2",TNoParam(),TNoParam())->Check = true;
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
    exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
    var_Item.GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup;
    exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
    exontrol.EXRIBBONLib.Item var_Item1 = var_Items1.Add("Check 1",null,null);
    var_Item1.Check = true;
    var_Item1.Checked = true;
    var_Items1.Add("Check 2",null,null).Check = true;
exribbon1.Refresh();

```

JScript/JavaScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var var_Item = var_Items.Add("",2,null);
    var_Item.GroupPopup = 3;
    var var_Items1 = var_Item.Items;
    var var_Item1 = var_Items1.Add("Check 1",null,null);
    var_Item1.Check = true;
    var_Item1.Checked = true;
    var_Items1.Add("Check 2",null,null).Check = true;
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            With .Add("",2)
                .GroupPopup = 3 ' GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
            With .Items
                With .Add("Check 1")
```

```

        .Check = True
        .Checked = True
    End With
    .Add("Check 2").Check = True
End With
End With
End With
.Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
var_Item.GroupPopup =
EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
EXRIBBONLib.GroupPopupEnum.exGroupPopup;
EXRIBBONLib.Items var_Items1 = var_Item.Items;
EXRIBBONLib.Item var_Item1 = var_Items1.Add("Check 1",null,null);
var_Item1.Check = true;
var_Item1.Checked = true;
var_Items1.Add("Check 2",null,null).Check = true;
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Item1,com_Item2,com_Items,com_Items1;
    anytype var_Item,var_Item1,var_Item2,var_Items,var_Items1;
    ;

    super();
}

```

```

var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("",COMVariant::createFromInt(2)); com_Item =
var_Item;
    com_Item.GroupPopup(3/*exNoGroupPopupFrame | exGroupPopup*/);
    var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
    var_Item1 = com_Items1.Add("Check 1"); com_Item1 = var_Item1;
    com_Item1.Check(true);
    com_Item1.Checked(true);
    var_Item2 = COM::createFromObject(com_Items1.Add("Check 2"));
com_Item2 = var_Item2;
    com_Item2.Check(true);
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        with Add('',TObject(2),Nil) do
        begin
            GroupPopup :=
Integer(EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib.GroupPopupEnum.exGroupPopup);
            with Items do
            begin
                with Add('Check 1',Nil,Nil) do
                begin
                    Check := True;
                    Checked := True;
                end;
                Add('Check 2',Nil,Nil).Check := True;
            end;
        end;
    end;
end;
end;
end;

```

```
Refresh();  
end
```

Delphi (standard)

```
with Ribbon1 do  
begin  
  with Items do  
  begin  
    with Add('',OleVariant(2),Null) do  
    begin  
      GroupPopup := Integer(EXRIBBONLib_TLB.exNoGroupPopupFrame) Or  
Integer(EXRIBBONLib_TLB.exGroupPopup);  
      with Items do  
      begin  
        with Add('Check 1',Null,Null) do  
        begin  
          Check := True;  
          Checked := True;  
        end;  
        Add('Check 2',Null,Null).Check := True;  
      end;  
    end;  
  end;  
  Refresh();  
end
```

VFP

```
with thisform.Ribbon1  
  with .Items  
    with Add('',2)  
      .GroupPopup = 3 && GroupPopupEnum.exNoGroupPopupFrame Or  
GroupPopupEnum.exGroupPopup  
      with .Items  
        with Add("Check 1")  
          .Check = .T.  
          .Checked = .T.
```

```

        endwhile
        .Add("Check 2").Check = .T.
    endwhile
endwith
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Item2,var_Items,var_Items1

oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
var_Items1 = var_Item.Items
var_Item1 = var_Items1.Add("Check 1")
var_Item1.Check = true
var_Item1.Checked = true
// var_Items1.Add("Check 2").Check = true
var_Item2 = var_Items1.Add("Check 2")
with (oRibbon)
    TemplateDef = [Dim var_Item2]
    TemplateDef = var_Item2
    Template = [var_Item2.Check = true]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Item2 as P
Dim var_Items as P
Dim var_Items1 as P

```

```

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 3 'exNoGroupPopupFrame + exGroupPopup
var_Items1 = var_Item.Items
var_Item1 = var_Items1.Add("Check 1")
var_Item1.Check = .t.
var_Item1.Checked = .t.
'var_Items1.Add("Check 2").Check = .t.
var_Item2 = var_Items1.Add("Check 2")
oRibbon.TemplateDef = "Dim var_Item2"
oRibbon.TemplateDef = var_Item2
oRibbon.Template = "var_Item2.Check = True"

oRibbon.Refresh()

```

Visual Objects

```

local var_Item,var_Item1 as Item
local var_Items,var_Items1 as Items

var_Items := oDCOCX_Exontrol1:Items
var_Item := var_Items:Add("",2,nil)
var_Item:GroupPopup := exNoGroupPopupFrame | exGroupPopup
var_Items1 := var_Item:Items
var_Item1 := var_Items1:Add("Check 1",nil,nil)
var_Item1:Check := true
var_Item1.Checked := true
var_Items1:Add("Check 2",nil,nil):Check := true
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Item1,var_Items,var_Items1
```



```

oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
var_Items1 = var_Item.Items
var_Item1 = var_Items1.Add("Check 1")
var_Item1.Check = true
var_Item1.Checked = true
var_Items1.Add("Check 2").Check = true
oRibbon.Refresh()

```

Visual DataFlex

```

Procedure OnCreate
  Forward Send OnCreate
  Variant voltems
  Get ComItems to voltems
  Handle holtems
  Get Create (RefClass(cComItems)) to holtems
  Set pvComObject of holtems to voltems
  Variant voltem
  Get ComAdd of holtems "" 2 Nothing to voltem
  Handle holtem
  Get Create (RefClass(cComItem)) to holtem
  Set pvComObject of holtem to voltem
  Set ComGroupPopup of holtem to (OLEexNoGroupPopupFrame +
OLEexGroupPopup)
  Variant voltems1
  Get ComItems of holtem to voltems1
  Handle holtems1
  Get Create (RefClass(cComItems)) to holtems1
  Set pvComObject of holtems1 to voltems1
  Variant voltem1
  Get ComAdd of holtems1 "Check 1" Nothing Nothing to voltem1
  Handle holtem1
  Get Create (RefClass(cComItem)) to holtem1

```

```

Set pvComObject of holtem1 to voltem1
Set ComCheck of holtem1 to True
Set ComChecked of holtem1 to True
Send Destroy to holtem1
Variant voltem2
Get ComAdd of holtems1 "Check 2" Nothing Nothing to voltem2
Handle holtem2
Get Create (RefClass(cComItem)) to holtem2
Set pvComObject of holtem2 to voltem2
Set ComCheck of holtem2 to True
Send Destroy to holtem2
Send Destroy to holtems1
Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem,oltem1
    LOCAL oltems,oltems1
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( „{100,100}, {640,480}„, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/

```

```
oRibbon:create(,, {10,60},{610,370} )
```

```
oltems := oRibbon:Items()
```

```
oltem := oltems:Add("",2)
```

```
oltem:GroupPopup := 3/*exNoGroupPopupFrame+exGroupPopup*/
```

```
oltems1 := oltem:Items()
```

```
oltem1 := oltems1:Add("Check 1")
```

```
oltem1:Check := .T.
```

```
oltem1:Checked := .T.
```

```
oltems1:Add("Check 2"):Check := .T.
```

```
oRibbon:Refresh()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Item.Checked as Boolean

Retrieves or sets a value that indicates the item's state.

Type	Description
Boolean	A Boolean expression that indicates whether the item is checked or unchecked.

The Checked property specifies whether the item is checked or un-checked. The [Check](#) property indicates whether the current item displays a check box. The [Radio](#) property specifies whether the item displays a radio-button. The [RadioGroup](#) property specifies a group of radio-buttons. A radio group allows a single radio-item to be checked inside. The [GetChecked](#) property gets a collection of checked items. The [GetUnchecked](#) property gets a collection of unchecked items. The [GetRadio](#) method gets a safe array with the radio-items being checked within a radio group. The [AllowToggleRadio](#) property on True, allows a radio button to set on zero (unchecked), if the user clicks twice the radio button. Usually, clicking a radio-button makes the previously checked radio-button in the same group, to be unchecked, and the newly clicked item to be checked. Now, if the AllowToggleRadio property is True, clicking again the radio-button, allows the radio-button to be un-checked, so allows a radio group to have no radio button checked. Use the [ShowCheckedAsSelected](#) property on True, to show the checked items as selected.

How can I add check-buttons?

VBA (MS Access, Excell...)

```
With Ribbon1
    With .Items
        With .Add("",2)
            .GroupPopup = 3 ' GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
        With .Items
            With .Add("Check 1")
                .Check = True
                .Checked = True
            End With
            .Add("Check 2").Check = True
        End With
    End With
End With
.Refresh
```

End With

VB6

```
With Ribbon1
  With .Items
    With .Add("",2)
      .GroupPopup = GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
    With .Items
      With .Add("Check 1")
        .Check = True
        .Checked = True
      End With
      .Add("Check 2").Check = True
    End With
  End With
End With
.Refresh
End With
```

VB.NET

```
With Exribbon1
  With .Items
    With .Add("",2)
      .GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup
    With .Items
      With .Add("Check 1")
        .Check = True
        .Checked = True
      End With
      .Add("Check 2").Check = True
    End With
  End With
End With
```

```
.Refresh()  
End With
```

VB.NET for /COM

```
With AxRibbon1  
    With .Items  
        With .Add("",2)  
            .GroupPopup = EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or  
EXRIBBONLib.GroupPopupEnum.exGroupPopup  
            With .Items  
                With .Add("Check 1")  
                    .Check = True  
                    .Checked = True  
                End With  
                .Add("Check 2").Check = True  
            End With  
        End With  
    End With  
    .Refresh()  
End With
```

C++

```
/*  
    Copy and paste the following directives to your header file as  
    it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control  
Library'  
  
    #import <ExRibbon.dll>  
    using namespace EXRIBBONLib;  
*/  
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-  
>GetControlUnknown();  
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();  
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"",long(2),vtMissing);  
    var_Item->  
>PutGroupPopup(EXRIBBONLib::GroupPopupEnum(EXRIBBONLib::exNoGroupPopupF
```

```

| EXRIBBONLib::exGroupPopup));
    EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();
    EXRIBBONLib::IItemPtr var_Item1 = var_Items1->Add(L"Check
1",vtMissing,vtMissing);
        var_Item1->PutCheck(VARIANT_TRUE);
        var_Item1->PutChecked(VARIANT_TRUE);
    var_Items1->Add(L"Check 2",vtMissing,vtMissing)-
> PutCheck(VARIANT_TRUE);
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    Exribbonlib_tlb::IItemPtr var_Item = var_Items->Add(L"",TVariant(2),TNoParam());
        var_Item->GroupPopup =
Exribbonlib_tlb::GroupPopupEnum::exNoGroupPopupFrame |
Exribbonlib_tlb::GroupPopupEnum::exGroupPopup;
    Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;
        Exribbonlib_tlb::IItemPtr var_Item1 = var_Items1->Add(L"Check
1",TNoParam(),TNoParam());
            var_Item1->Check = true;
            var_Item1->Checked = true;
        var_Items1->Add(L"Check 2",TNoParam(),TNoParam())->Check = true;
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
    exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
        var_Item.GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup;
    exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
        exontrol.EXRIBBONLib.Item var_Item1 = var_Items1.Add("Check 1",null,null);
            var_Item1.Check = true;
            var_Item1.Checked = true;

```

```
var_Items1.Add("Check 2",null,null).Check = true;  
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    var var_Items = Ribbon1.Items;  
    var var_Item = var_Items.Add("",2,null);  
    var_Item.GroupPopup = 3;  
    var var_Items1 = var_Item.Items;  
    var var_Item1 = var_Items1.Add("Check 1",null,null);  
    var_Item1.Check = true;  
    var_Item1.Checked = true;  
    var_Items1.Add("Check 2",null,null).Check = true;  
    Ribbon1.Refresh();  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
    With Ribbon1  
        With .Items  
            With .Add("",2)
```



```
.GroupPopup = 3 ' GroupPopupEnum.exNoGroupPopupFrame Or  
GroupPopupEnum.exGroupPopup
```

```
With .Items
```

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

```
.Check = True
```

```
.Checked = True
```

```
End With
```

```
.Add("Check 2").Check = True
```

```
End With
```

```
End With
```

```
End With
```

```
.Refresh
```

```
End With
```

```
End Function
```

```
</SCRIPT>
```

```
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;  
EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);  
var_Item.GroupPopup =  
EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |  
EXRIBBONLib.GroupPopupEnum.exGroupPopup;  
EXRIBBONLib.Items var_Items1 = var_Item.Items;  
EXRIBBONLib.Item var_Item1 = var_Items1.Add("Check 1",null,null);  
var_Item1.Check = true;  
var_Item1.Checked = true;  
var_Items1.Add("Check 2",null,null).Check = true;  
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()  
{  
    COM com_Item,com_Item1,com_Item2,com_Items,com_Items1;
```

```

anytype var_Item,var_Item1,var_Item2,var_Items,var_Items1;
;

super();

var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("",COMVariant::createFromInt(2)); com_Item =
var_Item;
    com_Item.GroupPopup(3/*exNoGroupPopupFrame | exGroupPopup*/);
    var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
    var_Item1 = com_Items1.Add("Check 1"); com_Item1 = var_Item1;
    com_Item1.Check(true);
    com_Item1.Checked(true);
    var_Item2 = COM::createFromObject(com_Items1.Add("Check 2"));
com_Item2 = var_Item2;
    com_Item2.Check(true);
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        with Add('',TObject(2),Nil) do
        begin
            GroupPopup :=
Integer(EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib.GroupPopupEnum.exGroupPopup);
            with Items do
            begin
                with Add('Check 1',Nil,Nil) do
                begin
                    Check := True;
                    Checked := True;
                end;
            end;
        end;
    end;
end;

```

```

        Add('Check 2',Nil,Nil).Check := True;
    end;
end;
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin
        with Add('',OleVariant(2),Null) do
        begin
            GroupPopup := Integer(EXRIBBONLib_TLB.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib_TLB.exGroupPopup);
            with Items do
            begin
                with Add('Check 1',Null,Null) do
                begin
                    Check := True;
                    Checked := True;
                end;
                Add('Check 2',Null,Null).Check := True;
            end;
        end;
    end;
end;
Refresh();
end

```

VFP

```

with thisform.Ribbon1
    with .Items
        with .Add("",2)
            .GroupPopup = 3 && GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup

```

```

with .Items
  with .Add("Check 1")
    .Check = .T.
    .Checked = .T.
  endwhile
  .Add("Check 2").Check = .T.
endwith
endwith
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Item2,var_Items,var_Items1

oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
var_Items1 = var_Item.Items
var_Item1 = var_Items1.Add("Check 1")
var_Item1.Check = true
var_Item1.Checked = true
// var_Items1.Add("Check 2").Check = true
var_Item2 = var_Items1.Add("Check 2")
with (oRibbon)
  TemplateDef = [Dim var_Item2]
  TemplateDef = var_Item2
  Template = [var_Item2.Check = true]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P

```

```

Dim var_Item1 as P
Dim var_Item2 as P
Dim var_Items as P
Dim var_Items1 as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 3 'exNoGroupPopupFrame + exGroupPopup
var_Items1 = var_Item.Items
var_Item1 = var_Items1.Add("Check 1")
var_Item1.Check = .t.
var_Item1.Checked = .t.
'var_Items1.Add("Check 2").Check = .t.
var_Item2 = var_Items1.Add("Check 2")
oRibbon.TemplateDef = "Dim var_Item2"
oRibbon.TemplateDef = var_Item2
oRibbon.Template = "var_Item2.Check = True"

oRibbon.Refresh()

```

Visual Objects

```

local var_Item,var_Item1 as IItem
local var_Items,var_Items1 as IItems

var_Items := oDCOCX_Exontrol1.Items
var_Item := var_Items.Add("",2,nil)
var_Item:GroupPopup := exNoGroupPopupFrame | exGroupPopup
var_Items1 := var_Item.Items
var_Item1 := var_Items1.Add("Check 1",nil,nil)
var_Item1:Check := true
var_Item1.Checked := true
var_Items1.Add("Check 2",nil,nil):Check := true
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Item1,var_Items,var_Items1
```

```
oRibbon = ole_1.Object
```

```
var_Items = oRibbon.Items
```

```
var_Item = var_Items.Add("",2)
```

```
var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
```

```
var_Items1 = var_Item.Items
```

```
var_Item1 = var_Items1.Add("Check 1")
```

```
var_Item1.Check = true
```

```
var_Item1.Checked = true
```

```
var_Items1.Add("Check 2").Check = true
```

```
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
```

```
Forward Send OnCreate
```

```
Variant voltems
```

```
Get ComItems to voltems
```

```
Handle holtems
```

```
Get Create (RefClass(cComItems)) to holtems
```

```
Set pvComObject of holtems to voltems
```

```
Variant voltem
```

```
Get ComAdd of holtems "" 2 Nothing to voltem
```

```
Handle holtem
```

```
Get Create (RefClass(cComItem)) to holtem
```

```
Set pvComObject of holtem to voltem
```

```
Set ComGroupPopup of holtem to (OLEexNoGroupPopupFrame +  
OLEexGroupPopup)
```

```
Variant voltems1
```

```
Get ComItems of holtem to voltems1
```

```
Handle holtems1
```

```
Get Create (RefClass(cComItems)) to holtems1
```

```
Set pvComObject of holtems1 to voltems1
```

```
Variant voltem1
```

```

Get ComAdd of holtems1 "Check 1" Nothing Nothing to voltem1
Handle holtem1
Get Create (RefClass(cComItem)) to holtem1
Set pvComObject of holtem1 to voltem1
    Set ComCheck of holtem1 to True
    Set ComChecked of holtem1 to True
Send Destroy to holtem1
Variant voltem2
Get ComAdd of holtems1 "Check 2" Nothing Nothing to voltem2
Handle holtem2
Get Create (RefClass(cComItem)) to holtem2
Set pvComObject of holtem2 to voltem2
    Set ComCheck of holtem2 to True
Send Destroy to holtem2
Send Destroy to holtems1
Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem,oltem1
    LOCAL oltems,oltems1
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( „{100,100}, {640,480},„ .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

```

```

oRibbon := XbpActiveXControl():new( oForm:drawingArea )
oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
oRibbon:create(, {10,60},{610,370} )

oltems := oRibbon:Items()
oltem := oltems:Add("",2)
oltem:GroupPopup := 3/*exNoGroupPopupFrame+exGroupPopup*/
oltems1 := oltem:Items()
oltem1 := oltems1:Add("Check 1")
oltem1:Check := .T.
oltem1:Checked := .T.
oltems1:Add("Check 2"):Check := .T.
oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
  nEvent := AppEvent( @mp1, @mp2, @oXbp )
  oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```


property Item.CloseOnClick as CloseOnClickEnum

Specifies the way the owner menu is closed once the user clicks the item.

Type	Description
CloseOnClickEnum	A CloseOnClickEnum expression that determines the way the hosting menu is closed when the user clicks the item.

You can specify a different way of closing the current item by specifying a different value for Item's CloseOnClick property. For instance, you can specify the Item's CloseOnClick property on exCloseOnClickOutside, for items of button type, so the hosting menu won't be closed when the user clicks the button, and so the user can click multiple times the item without closing the menu. The [ShowAsButton](#) property indicates whether the item should look as a button. The [ShowLocalPopup](#) property specifies whether the item's popup is shown as local. Clicking any item inside a local popup makes the popup itself to close including all its descendent sub-menus, without closing any ascendant sub-menus.

property Item.Cursor as Variant

Specifies the shape of the cursor when mouse hovers the item.

Type	Description
Variant	A String expression that defines the cursor to be shown when the cursor hovers the item. The Valid values are listed bellow. Also the Cursor property could point to a cursor file to be loaded and shown while the cursor hovers the item.

By default, the Cursor property is "exDefault". Use the Cursor property of the Item object to specify a different cursor when it hovers the item only. Use the [Cursor](#) property to specify a different cursor when it hovers the ribbon control.

The supported values are:

- "exDefault", Standard arrow
- "exArrow", Standard arrow
- "exCross", Crosshair
- "exIBeam", I-beam
- "exIcon", Reserved
- "exSize", Reserved, use the "exSizeAll"
- "exSizeNESW", Double-pointed arrow pointing northeast and southwest
- "exSizeNS", Double-pointed arrow pointing north and south
- "exSizeNWSE", Double-pointed arrow pointing northwest and southeast
- "exSizeWE", Double-pointed arrow pointing west and east
- "exUpArrow", Vertical arrow
- "exHourglass", Hourglass
- "exNoDrop", Slashed circle
- "exArrowHourglass"
- "exHelp", Arrow and question mark
- "exSizeAll", Four-pointed arrow pointing north, south, east, and west
- "exHand", Hand

Any other value indicates the path to a cursor file to be displayed when the pointer hovers the ribbon control/item.

Can I change the cursor where it hovers the item?

VBA (MS Access, Excell...)

With Ribbon1

.Cursor = "exCross"

```
With .Items
    .Add "Item 1"
    .Add("Item 2").Cursor = "exNoDrop"
End With
.Refresh
End With
```

VB6

```
With Ribbon1
    .Cursor = "exCross"
    With .Items
        .Add "Item 1"
        .Add("Item 2").Cursor = "exNoDrop"
    End With
    .Refresh
End With
```

VB.NET

```
With Exribbon1
    .Cursor = "exCross"
    With .Items
        .Add("Item 1")
        .Add("Item 2").Cursor = "exNoDrop"
    End With
    .Refresh()
End With
```

VB.NET for /COM

```
With AxRibbon1
    .Cursor = "exCross"
    With .Items
        .Add("Item 1")
        .Add("Item 2").Cursor = "exNoDrop"
    End With
    .Refresh()
```

C++

```

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
spRibbon1->PutCursor("exCross");
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    var_Items->Add(L"Item 1",vtMissing,vtMissing);
    var_Items->Add(L"Item 2",vtMissing,vtMissing)->PutCursor("exNoDrop");
spRibbon1->Refresh();

```

C++ Builder

```

Ribbon1->set_Cursor(TVariant("exCross"));
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    var_Items->Add(L"Item 1",TNoParam(),TNoParam());
    var_Items->Add(L"Item 2",TNoParam(),TNoParam())-
>set_Cursor(TVariant("exNoDrop"));
Ribbon1->Refresh();

```

C#

```

exribbon1.Cursor = "exCross";
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
    var_Items.Add("Item 1",null,null);
    var_Items.Add("Item 2",null,null).Cursor = "exNoDrop";
exribbon1.Refresh();

```

JScript/JavaScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Ribbon1.Cursor = "exCross";
    var var_Items = Ribbon1.Items;
    var_Items.Add("Item 1",null,null);
    var_Items.Add("Item 2",null,null).Cursor = "exNoDrop";
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        .Cursor = "exCross"
        With .Items
            .Add "Item 1"
            .Add("Item 2").Cursor = "exNoDrop"
        End With
        .Refresh
    End With
End Function
```

```
</SCRIPT>  
</BODY>
```

C# for /COM

```
axRibbon1.Cursor = "exCross";  
EXRIBBONLib.Items var_Items = axRibbon1.Items;  
    var_Items.Add("Item 1",null,null);  
    var_Items.Add("Item 2",null,null).Cursor = "exNoDrop";  
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()  
{  
    COM com_Item,com_Items;  
    anytype var_Item,var_Items;  
    ;  
  
    super();  
  
    exribbon1.Cursor("exCross");  
    var_Items = exribbon1.Items(); com_Items = var_Items;  
        com_Items.Add("Item 1");  
        var_Item = COM::createFromObject(com_Items.Add("Item 2")); com_Item =  
var_Item;  
        com_Item.Cursor("exNoDrop");  
    exribbon1.Refresh();  
}
```

Delphi 8 (.NET only)

```
with AxRibbon1 do  
begin  
    Cursor := 'exCross';  
    with Items do
```

```
begin
  Add('Item 1',Nil,Nil);
  Add('Item 2',Nil,Nil).Cursor := 'exNoDrop';
end;
Refresh();
end
```

Delphi (standard)

```
with Ribbon1 do
begin
  Cursor := 'exCross';
  with Items do
  begin
    Add('Item 1',Null,Null);
    Add('Item 2',Null,Null).Cursor := 'exNoDrop';
  end;
  Refresh();
end
```

VFP

```
with thisform.Ribbon1
.Cursor = "exCross"
with .Items
.Add("Item 1")
.Add("Item 2").Cursor = "exNoDrop"
endwith
.Refresh
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Items

oRibbon = form.ActiveX1.nativeObject
oRibbon.Cursor = "exCross"
var_Items = oRibbon.Items
```

```

var_Items.Add("Item 1")
// var_Items.Add("Item 2").Cursor = "exNoDrop"
var_Item = var_Items.Add("Item 2")
with (oRibbon)
    TemplateDef = [Dim var_Item]
    TemplateDef = var_Item
    Template = [var_Item.Cursor = "exNoDrop"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
oRibbon.Cursor = "exCross"
var_Items = oRibbon.Items
var_Items.Add("Item 1")
' var_Items.Add("Item 2").Cursor = "exNoDrop"
var_Item = var_Items.Add("Item 2")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.Cursor = \"exNoDrop\""

oRibbon.Refresh()

```

Visual Objects

```

local var_Items as IItems

oDCOCX_Exontrol1:Cursor := "exCross"
var_Items := oDCOCX_Exontrol1.Items
var_Items.Add("Item 1",nil,nil)
var_Items.Add("Item 2",nil,nil):Cursor := "exNoDrop"

```



```
oDCOCX_Exontrol1:Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Items  
  
oRibbon = ole_1.Object  
oRibbon.Cursor = "exCross"  
var_Items = oRibbon.Items  
    var_Items.Add("Item 1")  
    var_Items.Add("Item 2").Cursor = "exNoDrop"  
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate  
    Forward Send OnCreate  
    Set ComCursor to "exCross"  
    Variant voltems  
    Get ComItems to voltems  
    Handle holtems  
    Get Create (RefClass(cComItems)) to holtems  
    Set pvComObject of holtems to voltems  
        Get ComAdd of holtems "Item 1" Nothing Nothing to Nothing  
        Variant voltem  
        Get ComAdd of holtems "Item 2" Nothing Nothing to voltem  
        Handle holtem  
        Get Create (RefClass(cComItem)) to holtem  
        Set pvComObject of holtem to voltem  
            Set ComCursor of holtem to "exNoDrop"  
        Send Destroy to holtem  
    Send Destroy to holtems  
    Send ComRefresh  
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
    LOCAL oForm
```

```
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
    LOCAL oltems
```

```
    LOCAL oRibbon
```

```
    oForm := XbpDialog():new( AppDesktop() )
```

```
    oForm:drawingArea:clipChildren := .T.
```

```
    oForm:create( ,, {100,100}, {640,480},, .F. )
```

```
    oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
```

```
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-  
CFBE431702E2}*/
```

```
    oRibbon:create(,, {10,60},{610,370} )
```

```
    oRibbon:Cursor := "exCross"
```

```
    oltems := oRibbon:Items()
```

```
        oltems:Add("Item 1")
```

```
        oltems:Add("Item 2"):Cursor := "exNoDrop"
```

```
    oRibbon:Refresh()
```

```
    oForm:Show()
```

```
    DO WHILE nEvent != xbeP_Quit
```

```
        nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
        oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
    ENDDO
```

```
RETURN
```

property Item.EditBorder as EditBorderEnum

Specifies the border for the inside edit control.

Type	Description
EditBorderEnum	An EditBorderEnum expression that specifies the border to be shown around the item's text box.

The EditBorder property specifies the border to be shown around the item's text box. The [EditCaption](#) property specifies the caption to be shown on the item's Edit text box. Use the [AllowEdit](#) property to add a text-box inside the item, so the user can type any characters inside. The [EditWidth](#) property specifies the width of the text-box inside the item. You can use the [Get](#) method to collect all items of Edit type. The [EditChange](#) event notifies your application once the user alters the item's text-box caption.

How can I change/hide the editor's border?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("Text")
      .AllowEdit = 1
      .EditBorder = -1
    End With
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    With .Add("Text")
      .AllowEdit = exItemEditText
      .EditBorder = exEditBorderInset
    End With
  End With
  .Refresh
End With
```

VB.NET

```
With Exribbon1
    With .Items
        With .Add("Text")
            .AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditText
            .EditBorder = exontrol.EXRIBBONLib.EditBorderEnum.exEditBorderInset
        End With
    End With
    .Refresh()
End With
```

VB.NET for /COM

```
With AxRibbon1
    With .Items
        With .Add("Text")
            .AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditText
            .EditBorder = EXRIBBONLib.EditBorderEnum.exEditBorderInset
        End With
    End With
    .Refresh()
End With
```

C++

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Text",vtMissing,vtMissing);
```

```
var_Item->PutAllowEdit(EXRIBBONLib::exItemEditText);  
var_Item->PutEditBorder(EXRIBBONLib::exEditBorderInset);  
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;  
Exribbonlib_tlb::IItemPtr var_Item = var_Items->  
>Add(L"Text",TNoParam(),TNoParam());  
var_Item->AllowEdit = Exribbonlib_tlb::AllowEditEnum::exItemEditText;  
var_Item->EditBorder = Exribbonlib_tlb::EditBorderEnum::exEditBorderInset;  
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Text",null,null);  
var_Item.AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditText;  
var_Item.EditBorder = exontrol.EXRIBBONLib.EditBorderEnum.exEditBorderInset;  
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
var var_Items = Ribbon1.Items;  
var var_Item = var_Items.Add("Text",null,null);  
var_Item.AllowEdit = 1;  
var_Item.EditBorder = -1;  
Ribbon1.Refresh();  
}
```

```
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
  With Ribbon1  
    With .Items  
      With .Add("Text")  
        .AllowEdit = 1  
        .EditBorder = -1  
      End With  
    End With  
    .Refresh  
  End With  
End Function  
</SCRIPT>  
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;  
EXRIBBONLib.Item var_Item = var_Items.Add("Text",null,null);  
var_Item.AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditText;  
var_Item.EditBorder = EXRIBBONLib.EditBorderEnum.exEditBorderInset;  
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("Text"); com_Item = var_Item;
    com_Item.AllowEdit(1/*exItemEditText*/);
    com_Item.EditBorder(-1/*exEditBorderInset*/);
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        with Add('Text',Nil,Nil) do
        begin
            AllowEdit := EXRIBBONLib.AllowEditEnum.exItemEditText;
            EditBorder := EXRIBBONLib.EditBorderEnum.exEditBorderInset;
        end;
    end;
    Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin
        with Add('Text',Null,Null) do
        begin

```

```

AllowEdit := EXRIBBONLib_TLB.exItemEditText;
EditBorder := EXRIBBONLib_TLB.exEditBorderInset;
end;
end;
Refresh();
end

```

VFP

```

with thisform.Ribbon1
  with .Items
    with .Add("Text")
      .AllowEdit = 1
      .EditBorder = -1
    endwith
  endwith
  .Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Items

oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("Text")
var_Item.AllowEdit = 1
var_Item.EditBorder = -1
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex

```



```
var_Items = oRibbon.Items
var_Item = var_Items.Add("Text")
var_Item.AllowEdit = 1
var_Item.EditBorder = -1
oRibbon.Refresh()
```

Visual Objects

```
local var_Item as IItem
local var_Items as IItems

var_Items := oDCOCX_Exontrol1.Items
var_Item := var_Items.Add("Text",nil,nil)
var_Item.AllowEdit := exItemEditText
var_Item.EditBorder := exEditBorderInset
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Items

oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Item = var_Items.Add("Text")
var_Item.AllowEdit = 1
var_Item.EditBorder = -1
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
    Variant voltems
    Get ComItems to voltems
    Handle holtems
```

```

Get Create (RefClass(cComItems)) to holtems
Set pvComObject of holtems to voltems
    Variant voltem
    Get ComAdd of holtems "Text" Nothing Nothing to voltem
    Handle holtem
    Get Create (RefClass(cComItem)) to holtem
    Set pvComObject of holtem to voltem
        Set ComAllowEdit of holtem to OLEexItemEditText
        Set ComEditBorder of holtem to OLEexEditBorderInset
    Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oltems := oRibbon:Items()

```

```
    oltem := oltems:Add("Text")
        oltem:AllowEdit := 1/*exItemEditText*/
        oltem:EditBorder := -1/*exEditBorderInset*/
oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN
```

property Item.EditCaption as String

Specifies the edit's caption when the item contains an edit control.

Type	Description
String	A String expression that specifies the caption to be shown on the item's text box.

The EditCaption property specifies the caption to be shown on the item's Edit text box. Use the [AllowEdit](#) property to add a text-box inside the item, so the user can type any characters inside. The [EditWidth](#) property specifies the width of the text-box inside the item. The [EditBorder](#) property specifies the border to be shown around the item's text box. You can use the [Get](#) method to collect all items of Edit type. The [EditChange](#) event notifies your application once the user alters the item's text-box caption. The [EditValue](#) property indicates the edit's value.

How can I change the edit's caption?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("Text")
      .AllowEdit = 1
      .EditCaption = "caption"
    End With
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    With .Add("Text")
      .AllowEdit = exlItemEditText
      .EditCaption = "caption"
    End With
  End With
  .Refresh
End With
```

VB.NET

```
With Exribbon1
    With .Items
        With .Add("Text")
            .AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditText
            .EditCaption = "caption"
        End With
    End With
    .Refresh()
End With
```

VB.NET for /COM

```
With AxRibbon1
    With .Items
        With .Add("Text")
            .AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditText
            .EditCaption = "caption"
        End With
    End With
    .Refresh()
End With
```

C++

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Text",vtMissing,vtMissing);
```

```
var_Item->PutAllowEdit(EXRIBBONLib::exItemEditText);  
var_Item->PutEditCaption(L"caption");  
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;  
Exribbonlib_tlb::IItemPtr var_Item = var_Items->  
Add(L"Text",TNoParam(),TNoParam());  
var_Item->AllowEdit = Exribbonlib_tlb::AllowEditEnum::exItemEditText;  
var_Item->EditCaption = L"caption";  
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Text",null,null);  
var_Item.AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditText;  
var_Item.EditCaption = "caption";  
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
var var_Items = Ribbon1.Items;  
var var_Item = var_Items.Add("Text",null,null);  
var_Item.AllowEdit = 1;  
var_Item.EditCaption = "caption";  
Ribbon1.Refresh();
```

```
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
  With Ribbon1  
    With .Items  
      With .Add("Text")  
        .AllowEdit = 1  
        .EditCaption = "caption"  
      End With  
    End With  
    .Refresh  
  End With  
End Function  
</SCRIPT>  
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;  
EXRIBBONLib.Item var_Item = var_Items.Add("Text",null,null);  
var_Item.AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditText;  
var_Item.EditCaption = "caption";  
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("Text"); com_Item = var_Item;
    com_Item.AllowEdit(1/*exItemEditText*/);
    com_Item.EditCaption("caption");
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        with Add('Text',Nil,Nil) do
        begin
            AllowEdit := EXRIBBONLib.AllowEditEnum.exItemEditText;
            EditCaption := 'caption';
        end;
    end;
    Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin
        with Add('Text',Null,Null) do
        begin

```



```

AllowEdit := EXRIBBONLib_TLB.exItemEditText;
EditCaption := 'caption';
end;
end;
Refresh();
end

```

VFP

```

with thisform.Ribbon1
  with .Items
    with .Add("Text")
      .AllowEdit = 1
      .EditCaption = "caption"
    endwith
  endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Items

oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("Text")
var_Item.AllowEdit = 1
var_Item.EditCaption = "caption"
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex

```

```
var_Items = oRibbon.Items
var_Item = var_Items.Add("Text")
var_Item.AllowEdit = 1
var_Item.EditCaption = "caption"
oRibbon.Refresh()
```

Visual Objects

```
local var_Item as IItem
local var_Items as IItems

var_Items := oDCOCX_Exontrol1.Items
var_Item := var_Items.Add("Text",nil,nil)
var_Item.AllowEdit := exItemEditText
var_Item.EditCaption := "caption"
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Items

oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Item = var_Items.Add("Text")
var_Item.AllowEdit = 1
var_Item.EditCaption = "caption"
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
    Variant voltems
    Get ComItems to voltems
    Handle holtems
```

```

Get Create (RefClass(cComItems)) to holtems
Set pvComObject of holtems to voltems
Variant voltem
Get ComAdd of holtems "Text" Nothing Nothing to voltem
Handle holtem
Get Create (RefClass(cComItem)) to holtem
Set pvComObject of holtem to voltem
Set ComAllowEdit of holtem to OLEexItemEditText
Set ComEditCaption of holtem to "caption"
Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oltems := oRibbon:Items()

```

```
    oltem := oltems:Add("Text")
    oltem:AllowEdit := 1/*exItemEditText*/
    oltem:EditCaption := "caption"
oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN
```

property Item.EditMask as String

Specifies the edit's mask when the item contains an masked edit control.

Type	Description
String	A string expression that indicates the mask of the edit's field.

By default, the EditMask property is "" (empty string, no masking). The EditMask property is valid for exItemEditMask editors. The [AllowEdit](#) property associates an editor to the current item. The EditMask property specifies the mask of the edit field. The [EditValue](#) property specifies the value of the edit field, without the masking characters. The [EditOption\(exEditMaskFloat\)](#) specifies whether the edit field mask a floating/decimal/integer point number. The EditMask property depends on the [EditOption\(exEditMaskFloat\)](#) value, as explained bellow.

A) If the [EditOption\(exEditMaskFloat\)](#) property is False (by default), the EditMask is defined such as:

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

!(999) 000 0000;1;;;select=1,empty,overtypewarning=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 *overtyp*e 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;overtime,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 use 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 **ť**(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)
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 overtyping option, it indicates that the user can toggle the insert/over-type mode using the Insert key. For instance, the "###:###;0;inserttype,overtyping", 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.
- **overtyping**, 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;overtyping,inserttype", indicates that the field enter in over-type mode, and insert-type mode is allowed. The "###:###;0;overtyping", 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.
- **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,overtyp"`, 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,overtyp,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

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.

B) If the [EditOption\(exEditMaskFloat\)](#) property is True, the EditMask is defined such as:

The EditMask property may indicate the followings:

- **negative number**: if the first character in the mask is - (minus) the control supports negative numbers. Pressing the - key will toggle the sign of the number. The + sign is never displayed.
- **decimal symbol**: the last character that's different than # (digit), or 0 (zero) indicates the decimal symbol. If it is not present the control mask a floating point number without decimals.
- **thousand symbol**: the thousand symbol is the last character that's not a # (digit), 0 (zero) or it is not the decimal symbol as explained earlier, if present.
- the maximum **number of decimals** in the number (the # or 0 character after the decimal symbol)
- the maximum number of digits in the integer part (the number of # or 0 character before decimal symbol)
- the **0** character indicates a **leading-zero**. The count of 0 (zero) characters before decimal character indicates the leading-zero for integer part of the control, while the count of 0 (zero) characters after the decimal separator indicates the leading-zero for decimal part of the control. For instance, the Mask on "-###,###,##0.00", while the control's Text property is 1, the control displays 1.00, if 1.1 if displays 1.10, and if empty, the 0.00 is displayed.

If the EditMask property is empty, the control takes the settings for the regional options like: Decimal Symbol , No. of digits after decimal, Digit grouping symbol.

Here are few samples:

The EditMask"-###.###.##0,00" filter floating point numbers a number for German settings ("," is the decimal sign, "." is the thousands separator). This format displays leading-zeros.

The EditMask"-###.###.###,##" filter floating point numbers a number for German settings ("," is the decimal sign, "." is the thousands separator)

The EditMask"-###,###,###.##" filter floating point numbers a number for English settings ("." is the decimal sign, "," is the thousands separator)

The EditMask"####" indicates a max-4 digit number (positive) without a decimal symbol and without digit grouping

The EditMask"-##.#" filters a floating point number from the -99.9 to 99.9 ("." is the decimal sign, no thousands separator)

The EditMask"#,###.##" filters a floating point number from the 0 to 9,999.99 with digit grouping ("." is the decimal sign, "," is the thousands separator).

property Item.EditOption(Option as EditOptionEnum) as Variant

Specifies different options for item's edit control.

Type	Description
Option as EditOptionEnum	An EditOptionEnum expression that specifies the option to be updated.
Variant	A VARIANT expression that indicates the option's value.

The EditOption property different options for item's edit control. The [AllowEdit](#) property associates an editor to the current item. The [EditCaption](#) property specifies the value to show in the edit field. The [EditWidth](#) property specifies the size/width of the edit field inside the item. The [EditBorder](#) property specifies whether the edit shows a border around it. The control fires the [EditChange](#) event when the user changes the edit's caption. For instance, the EditOption(exEditSpinStep) property specifies the step to advance when user clicks the editor's spin.

How can I assign a slider field to the item (method 2)?

VBA (MS Access, Excell...)

With Ribbon1

With .Items

With .Add("Slider")

.AllowEdit = 3

.EditWidth = -128

.EditValue = 25

End With

With .Add("Slider")

.AllowEdit = 3

.EditBorder = 0

.EditWidth = -128

.EditOption(1) = 50

.EditOption(2) = 450

.EditOption(3) = 2

.EditOption(4) = 50

.EditOption(5) = "value = vmin ? '
 ' + value : (value = vmax ?

'
 ' + value : (value = " & _

" 200 ? '
 <fgcolor FF0000>' + value : ")"

.EditValue = 345

```
End With
End With
.Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    With .Add("Slider")
      .AllowEdit = exItemEditSlider
      .EditWidth = -128
      .EditValue = 25
    End With
    With .Add("Slider")
      .AllowEdit = exItemEditSlider
      .EditBorder = exEditBorderNone
      .EditWidth = -128
      .EditOption(exEditMinValue) = 50
      .EditOption(exEditMaxValue) = 450
      .EditOption(exEditTickStyle) = 2
      .EditOption(exEditTickFrequency) = 50
      .EditOption(exEditTickLabel) = "value = vmin ? '<br> <font ;6> <b>' + value : (
value = vmax ? '<br> <font ;6> <b>' + value : ( value = " & _
" 200 ? '<br> <font ;6> <b> <fgcolor FF0000>' + value : " ) )"
      .EditValue = 345
    End With
  End With
  .Refresh
End With
```

VB.NET

```
With Exribbon1
  With .Items
    With .Add("Slider")
      .AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditSlider
      .EditWidth = -128
    End With
  End With
  .Refresh
End With
```

.EditValue = 25

End With

With .Add("Slider")

.AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditSlider

.EditBorder = exontrol.EXRIBBONLib.EditBorderEnum.exEditBorderNone

.EditWidth = -128

.set_EditOption(exontrol.EXRIBBONLib.EditOptionEnum.exEditMinValue,50)

.set_EditOption(exontrol.EXRIBBONLib.EditOptionEnum.exEditMaxValue,450)

.set_EditOption(exontrol.EXRIBBONLib.EditOptionEnum.exEditTickStyle,2)

.set_EditOption(exontrol.EXRIBBONLib.EditOptionEnum.exEditTickFrequency,50)

.set_EditOption(exontrol.EXRIBBONLib.EditOptionEnum.exEditTickLabel,"value

= vmin ? '
'+value : (value = vmax ? '
'+value : (

value = " & _

" 200 ? '
<fgcolor FF0000>'+value : "))")

.EditValue = 345

End With

End With

.Refresh()

End With

VB.NET for /COM

With AxRibbon1

With .Items

With .Add("Slider")

.AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditSlider

.EditWidth = -128

.EditValue = 25

End With

With .Add("Slider")

.AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditSlider

.EditBorder = EXRIBBONLib.EditBorderEnum.exEditBorderNone

.EditWidth = -128

.EditOption(EXRIBBONLib.EditOptionEnum.exEditMinValue) = 50

.EditOption(EXRIBBONLib.EditOptionEnum.exEditMaxValue) = 450

.EditOption(EXRIBBONLib.EditOptionEnum.exEditTickStyle) = 2

```

.EditOption(EXRIBBONLib.EditOptionEnum.exEditTickFrequency) = 50
.EditOption(EXRIBBONLib.EditOptionEnum.exEditTickLabel) = "value = vmin ?
'<br><font ;6><b>' + value : ( value = vmax ? '<br><font ;6><b>' + value : ( value = "
& _
" 200 ? '<br><font ;6><b><fgcolor FF0000>' + value : " ) )"
.EditValue = 345
End With
End With
.Refresh()
End With

```

C++

```

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

#import <ExRibbon.dll>
using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Slider",vtMissing,vtMissing);
var_Item->PutAllowEdit(EXRIBBONLib::exItemEditSlider);
var_Item->PutEditWidth(-128);
var_Item->PutEditValue(long(25));
EXRIBBONLib::IItemPtr var_Item1 = var_Items->Add(L"Slider",vtMissing,vtMissing);
var_Item1->PutAllowEdit(EXRIBBONLib::exItemEditSlider);
var_Item1->PutEditBorder(EXRIBBONLib::exEditBorderNone);
var_Item1->PutEditWidth(-128);
var_Item1->PutEditOption(EXRIBBONLib::exEditMinValue,long(50));
var_Item1->PutEditOption(EXRIBBONLib::exEditMaxValue,long(450));
var_Item1->PutEditOption(EXRIBBONLib::exEditTickStyle,long(2));
var_Item1->PutEditOption(EXRIBBONLib::exEditTickFrequency,long(50));
var_Item1->PutEditOption(EXRIBBONLib::exEditTickLabel,_bstr_t("value = vmin ?

```

```
'<br><font ;6><b>' + value : ( value = vmax ? '<br><font ;6><b>' + value : ( value
= ") +
" 200 ? '<br><font ;6><b><fgcolor FF0000>' + value : " ) )");
    var_Item1->PutEditValue(long(345));
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    Exribbonlib_tlb::IItemPtr var_Item = var_Items-
>Add(L"Slider",TNoParam(),TNoParam());
    var_Item->AllowEdit = Exribbonlib_tlb::AllowEditEnum::exItemEditSlider;
    var_Item->EditWidth = -128;
    var_Item->set_EditValue(TVariant(25));
    Exribbonlib_tlb::IItemPtr var_Item1 = var_Items-
>Add(L"Slider",TNoParam(),TNoParam());
    var_Item1->AllowEdit = Exribbonlib_tlb::AllowEditEnum::exItemEditSlider;
    var_Item1->EditBorder = Exribbonlib_tlb::EditBorderEnum::exEditBorderNone;
    var_Item1->EditWidth = -128;
    var_Item1-
>set_EditOption(Exribbonlib_tlb::EditOptionEnum::exEditMinValue,TVariant(50));
    var_Item1-
>set_EditOption(Exribbonlib_tlb::EditOptionEnum::exEditMaxValue,TVariant(450));
    var_Item1-
>set_EditOption(Exribbonlib_tlb::EditOptionEnum::exEditTickStyle,TVariant(2));
    var_Item1-
>set_EditOption(Exribbonlib_tlb::EditOptionEnum::exEditTickFrequency,TVariant(50));
    var_Item1-
>set_EditOption(Exribbonlib_tlb::EditOptionEnum::exEditTickLabel,TVariant(String("valu
= vmin ? '<br><font ;6><b>' + value : ( value = vmax ? '<br><font ;6><b>' + value :
( value = ") +
" 200 ? '<br><font ;6><b><fgcolor FF0000>' + value : " ) )"));
    var_Item1->set_EditValue(TVariant(345));
Ribbon1->Refresh();
```

C#


```

exontrol.EXRIBBONLib.Items var_Item = exribbon1.Items;
    exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Slider",null,null);
        var_Item.AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditSlider;
        var_Item.EditWidth = -128;
        var_Item.EditValue = 25;
    exontrol.EXRIBBONLib.Item var_Item1 = var_Items.Add("Slider",null,null);
        var_Item1.AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditSlider;
        var_Item1.EditBorder =
exontrol.EXRIBBONLib.EditBorderEnum.exEditBorderNone;
        var_Item1.EditWidth = -128;

var_Item1.set_EditOption(exontrol.EXRIBBONLib.EditOptionEnum.exEditMinValue,50);

var_Item1.set_EditOption(exontrol.EXRIBBONLib.EditOptionEnum.exEditMaxValue,450);

var_Item1.set_EditOption(exontrol.EXRIBBONLib.EditOptionEnum.exEditTickStyle,2);

var_Item1.set_EditOption(exontrol.EXRIBBONLib.EditOptionEnum.exEditTickFrequency,

var_Item1.set_EditOption(exontrol.EXRIBBONLib.EditOptionEnum.exEditTickLabel,"value
= vmin ? '<br><font ;6><b>' +value : ( value = vmax ? '<br><font ;6><b>' +value :
( value = " +
" 200 ? '<br><font ;6><b><fgcolor FF0000>' +value : " ) )");
        var_Item1.EditValue = 345;
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()

```

```

{
var var_Items = Ribbon1.Items;
var var_Item = var_Items.Add("Slider",null,null);
var_Item.AllowEdit = 3;
var_Item.EditWidth = -128;
var_Item.EditValue = 25;
var var_Item1 = var_Items.Add("Slider",null,null);
var_Item1.AllowEdit = 3;
var_Item1.EditBorder = 0;
var_Item1.EditWidth = -128;
var_Item1.EditOption(1) = 50;
var_Item1.EditOption(2) = 450;
var_Item1.EditOption(3) = 2;
var_Item1.EditOption(4) = 50;
var_Item1.EditOption(5) = "value = vmin ? ' <br> <font ;6> <b>' + value : (
value = vmax ? ' <br> <font ;6> <b>' + value : ( value = " +
" 200 ? ' <br> <font ;6> <b> <fgcolor FF0000>' + value : " ) )";
var_Item1.EditValue = 345;
Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
With Ribbon1
With .Items
With .Add("Slider")
.AllowEdit = 3
.EditWidth = -128

```

```

        .EditValue = 25
    End With
    With .Add("Slider")
        .AllowEdit = 3
        .EditBorder = 0
        .EditWidth = -128
        .EditOption(1) = 50
        .EditOption(2) = 450
        .EditOption(3) = 2
        .EditOption(4) = 50
        .EditOption(5) = "value = vmin ? '<br><font ;6><b>' + value : ( value =
vmax ? '<br><font ;6><b>' + value : ( value =" & _
" 200 ? '<br><font ;6><b><fgcolor FF0000>' + value : " ) )"
        .EditValue = 345
    End With
End With
.Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
EXRIBBONLib.Item var_Item = var_Items.Add("Slider",null,null);
var_Item.AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditSlider;
var_Item.EditWidth = -128;
var_Item.EditValue = 25;
EXRIBBONLib.Item var_Item1 = var_Items.Add("Slider",null,null);
var_Item1.AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditSlider;
var_Item1.EditBorder = EXRIBBONLib.EditBorderEnum.exEditBorderNone;
var_Item1.EditWidth = -128;
var_Item1.set_EditOption(EXRIBBONLib.EditOptionEnum.exEditMinValue,50);
var_Item1.set_EditOption(EXRIBBONLib.EditOptionEnum.exEditMaxValue,450);
var_Item1.set_EditOption(EXRIBBONLib.EditOptionEnum.exEditTickStyle,2);

```

```

var_Item1.set_EditOption(EXRIBBONLib.EditOptionEnum.exEditTickFrequency,50);
    var_Item1.set_EditOption(EXRIBBONLib.EditOptionEnum.exEditTickLabel,"value
= vmin ? '<br><font ;6><b>' +value : ( value = vmax ? '<br><font ;6><b>' +value : (
value =" +
" 200 ? '<br><font ;6><b><fgcolor FF0000>' +value : " ) )");
    var_Item1.EditValue = 345;
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Item1,com_Items;
    anytype var_Item,var_Item1,var_Items;
    str var_s;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("Slider"); com_Item = var_Item;
    com_Item.AllowEdit(3/*exItemEditSlider*/);
    com_Item.EditWidth(-128);
    com_Item.EditValue(COMVariant::createFromInt(25));
    var_Item1 = com_Items.Add("Slider"); com_Item1 = var_Item1;
    com_Item1.AllowEdit(3/*exItemEditSlider*/);
    com_Item1.EditBorder(0/*exEditBorderNone*/);
    com_Item1.EditWidth(-128);
    com_Item1.EditOption(1/*exEditMinValue*/,COMVariant::createFromInt(50));
    com_Item1.EditOption(2/*exEditMaxValue*/,COMVariant::createFromInt(450));
    com_Item1.EditOption(3/*exEditTickStyle*/,COMVariant::createFromInt(2));

    com_Item1.EditOption(4/*exEditTickFrequency*/,COMVariant::createFromInt(50));
    var_s = "value = vmin ? '<br><font ;6><b>' +value : ( value = vmax ? '<br>
<font ;6><b>' +value : ( value = ";

```

```

var_s = var_s + "200 ? ' <br> <font ;6> <b> <fgcolor FF0000>' +value : " ) )";
com_Item1.EditOption(5/*exEditTickLabel*/,COMVariant::createFromStr(var_s));
com_Item1.EditValue(COMVariant::createFromInt(345));
exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
  with Items do
  begin
    with Add('Slider',Nil,Nil) do
    begin
      AllowEdit := EXRIBBONLib.AllowEditEnum.exItemEditSlider;
      EditWidth := -128;
      EditValue := TObject(25);
    end;
    with Add('Slider',Nil,Nil) do
    begin
      AllowEdit := EXRIBBONLib.AllowEditEnum.exItemEditSlider;
      EditBorder := EXRIBBONLib.EditBorderEnum.exEditBorderNone;
      EditWidth := -128;
      EditOption[EXRIBBONLib.EditOptionEnum.exEditMinValue] := TObject(50);
      EditOption[EXRIBBONLib.EditOptionEnum.exEditMaxValue] := TObject(450);
      EditOption[EXRIBBONLib.EditOptionEnum.exEditTickStyle] := TObject(2);
      EditOption[EXRIBBONLib.EditOptionEnum.exEditTickFrequency] :=
TObject(50);
      EditOption[EXRIBBONLib.EditOptionEnum.exEditTickLabel] := 'value = vmin ?
"<br> <font ;6> <b>' +value : ( value = vmax ? "<br> <font ;6> <b>' +value : ( value
= ' +
'200 ? "<br> <font ;6> <b> <fgcolor FF0000>' +value : "" ) )';
      EditValue := TObject(345);
    end;
  end;
  Refresh();
end

```

Delphi (standard)

```
with Ribbon1 do
begin
  with Items do
  begin
    with Add('Slider',Null,Null) do
    begin
      AllowEdit := EXRIBBONLib_TLB.exItemEditSlider;
      EditWidth := -128;
      EditValue := OleVariant(25);
    end;
    with Add('Slider',Null,Null) do
    begin
      AllowEdit := EXRIBBONLib_TLB.exItemEditSlider;
      EditBorder := EXRIBBONLib_TLB.exEditBorderNone;
      EditWidth := -128;
      EditOption[EXRIBBONLib_TLB.exEditMinValue] := OleVariant(50);
      EditOption[EXRIBBONLib_TLB.exEditMaxValue] := OleVariant(450);
      EditOption[EXRIBBONLib_TLB.exEditTickStyle] := OleVariant(2);
      EditOption[EXRIBBONLib_TLB.exEditTickFrequency] := OleVariant(50);
      EditOption[EXRIBBONLib_TLB.exEditTickLabel] := 'value = vmin ? "<br><font
;6><b>" + value : ( value = vmax ? "<br><font ;6><b>" + value : ( value = ' +
'200 ? "<br><font ;6><b><fgcolor FF0000>" + value : "" ) )';
      EditValue := OleVariant(345);
    end;
  end;
  Refresh();
end
```

VFP

```
with thisform.Ribbon1
  with .Items
    with .Add("Slider")
      .AllowEdit = 3
      .EditWidth = -128
      .EditValue = 25
```

```

endwith
with .Add("Slider")
    .AllowEdit = 3
    .EditBorder = 0
    .EditWidth = -128
    .EditOption(1) = 50
    .EditOption(2) = 450
    .EditOption(3) = 2
    .EditOption(4) = 50
    var_s = "value = vmin ? '<br><font ;6><b>' + value : ( value = vmax ? '<br><font ;6><b>' + value : ( value = "
    var_s = var_s + "200 ? '<br><font ;6><b><fgcolor FF0000>' + value : " ) )"
    .EditOption(5) = var_s
    .EditValue = 345
endwith
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Items

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("Slider")
    var_Item.AllowEdit = 3
    var_Item.EditWidth = -128
    var_Item.EditValue = 25
var_Item1 = var_Items.Add("Slider")
    var_Item1.AllowEdit = 3
    var_Item1.EditBorder = 0
    var_Item1.EditWidth = -128
    // var_Item1.EditOption(1) = 50
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1

```

```

    Template = [var_Item1.EditOption(1) = 50]
endwith
// var_Item1.EditOption(2) = 450
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.EditOption(2) = 450]
endwith
// var_Item1.EditOption(3) = 2
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.EditOption(3) = 2]
endwith
// var_Item1.EditOption(4) = 50
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.EditOption(4) = 50]
endwith
// var_Item1.EditOption(5) = "value = vmin ? '<br><font ;6><b>' + value :
( value = vmax ? '<br><font ;6><b>' + value : ( value = 200 ? '<br><font ;6>
<b><fgcolor FF0000>' + value : '' ) )"
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.EditOption(5) = "value = vmin ? '<br><font ;6>
<b>' + value : ( value = vmax ? '<br><font ;6><b>' + value : ( value = 200 ? '<br>
<font ;6><b><fgcolor FF0000>' + value : '' ) )"]
endwith
var_Item1.EditValue = 345
oRibbon.Refresh()

```

XBasic (Alpha Five)

Dim oRibbon as P


```
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P
```

```
oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
```

```
var_Item = var_Items.Add("Slider")
```

```
var_Item.AllowEdit = 3
```

```
var_Item.EditWidth = -128
```

```
var_Item.EditValue = 25
```

```
var_Item1 = var_Items.Add("Slider")
```

```
var_Item1.AllowEdit = 3
```

```
var_Item1.EditBorder = 0
```

```
var_Item1.EditWidth = -128
```

```
' var_Item1.EditValue = 50
```

```
oRibbon.TemplateDef = "Dim var_Item1"
```

```
oRibbon.TemplateDef = var_Item1
```

```
oRibbon.Template = "var_Item1.EditValue = 50"
```

```
' var_Item1.EditValue = 450
```

```
oRibbon.TemplateDef = "Dim var_Item1"
```

```
oRibbon.TemplateDef = var_Item1
```

```
oRibbon.Template = "var_Item1.EditValue = 450"
```

```
' var_Item1.EditValue = 2
```

```
oRibbon.TemplateDef = "Dim var_Item1"
```

```
oRibbon.TemplateDef = var_Item1
```

```
oRibbon.Template = "var_Item1.EditValue = 2"
```

```
' var_Item1.EditValue = 50
```

```
oRibbon.TemplateDef = "Dim var_Item1"
```

```
oRibbon.TemplateDef = var_Item1
```

```
oRibbon.Template = "var_Item1.EditValue = 50"
```

```
' var_Item1.EditValue = "value = vmin ? '<br><font ;6><b>' + value : (
value = vmax ? '<br><font ;6><b>' + value : ( value = 200 ? '<br><font ;6>
<b><fgcolor FF0000>' + value : '' ) )"
```

```

oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1
oRibbon.Template = "var_Item1.EditOption(5) = \"value = vmin ? '<br><font ;6><b>'+value : ( value = vmax ? '<br><font ;6><b>'+value : ( value = 200 ? '<br><font ;6><b><fgcolor FF0000>'+value : ' ' ) )\"

var_Item1.EditValue = 345
oRibbon.Refresh()

```

Visual Objects

```

local var_Item,var_Item1 as Item
local var_Items as Items

var_Items := oDCOCX_Exontrol1.Items
var_Item := var_Items.Add("Slider",nil,nil)
var_Item.AllowEdit := exItemEditSlider
var_Item.EditWidth := -128
var_Item.EditValue := 25
var_Item1 := var_Items.Add("Slider",nil,nil)
var_Item1.AllowEdit := exItemEditSlider
var_Item1.EditBorder := exEditBorderNone
var_Item1.EditWidth := -128
var_Item1:[EditOption,exEditMinValue] := 50
var_Item1:[EditOption,exEditMaxValue] := 450
var_Item1:[EditOption,exEditTickStyle] := 2
var_Item1:[EditOption,exEditTickFrequency] := 50
var_Item1:[EditOption,exEditTickLabel] := "value = vmin ? '<br><font ;6><b>'+value : ( value = vmax ? '<br><font ;6><b>'+value : ( value = 200 ? '<br><font ;6><b><fgcolor FF0000>'+value : ' ' ) )"
var_Item1.EditValue := 345
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Item1,var_Items
```

```

oRibbon = ole_1.Object
var_Items = oRibbon.Items
  var_Item = var_Items.Add("Slider")
    var_Item.AllowEdit = 3
    var_Item.EditWidth = -128
    var_Item.EditValue = 25
var_Item1 = var_Items.Add("Slider")
  var_Item1.AllowEdit = 3
  var_Item1.EditBorder = 0
  var_Item1.EditWidth = -128
  var_Item1.EditOption(1,50)
  var_Item1.EditOption(2,450)
  var_Item1.EditOption(3,2)
  var_Item1.EditOption(4,50)
  var_Item1.EditOption(5,"value = vmin ? '<br><font ;6><b>' +value : ( value =
vmax ? '<br><font ;6><b>' +value : ( value = 200 ? '<br><font ;6><b><fgcolor
FF0000>' +value : " ) )")
    var_Item1.EditValue = 345
oRibbon.Refresh()

```

Visual DataFlex

```

Procedure OnCreate
  Forward Send OnCreate
  Variant voltems
  Get Comltems to voltems
  Handle holtems
  Get Create (RefClass(cComltems)) to holtems
  Set pvComObject of holtems to voltems
  Variant voltem
  Get ComAdd of holtems "Slider" Nothing Nothing to voltem
  Handle holtem
  Get Create (RefClass(cComltem)) to holtem
  Set pvComObject of holtem to voltem
  Set ComAllowEdit of holtem to OLExltemEditSlider

```

```

Set ComEditWidth of holtem to -128
Set ComEditValue of holtem to 25
Send Destroy to holtem
Variant voltem1
Get ComAdd of holtems "Slider" Nothing Nothing to voltem1
Handle holtem1
Get Create (RefClass(cComItem)) to holtem1
Set pvComObject of holtem1 to voltem1
Set ComAllowEdit of holtem1 to OLEexItemEditSlider
Set ComEditBorder of holtem1 to OLEexEditBorderNone
Set ComEditWidth of holtem1 to -128
Set ComEditOption of holtem1 OLEexEditMinValue to 50
Set ComEditOption of holtem1 OLEexEditMaxValue to 450
Set ComEditOption of holtem1 OLEexEditTickStyle to 2
Set ComEditOption of holtem1 OLEexEditTickFrequency to 50
Set ComEditOption of holtem1 OLEexEditTickLabel to "value = vmin ? '<br>
<font ;6> <b>' + value : ( value = vmax ? '<br> <font ;6> <b>' + value : ( value = 200 ?
'<br> <font ;6> <b> <fgcolor FF0000>' + value : " ) )"
Set ComEditValue of holtem1 to 345
Send Destroy to holtem1
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem, oltem1
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )

```

```

oForm:drawingArea:clipChildren := .T.
oForm:create( ,, {100,100}, {640,480},,, .F. )
oForm:close := {|| PostAppEvent( xbeP_Quit )}

oRibbon := XbpActiveXControl():new( oForm:drawingArea )
oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
oRibbon:create(,, {10,60},{610,370} )

oltems := oRibbon:Items()
oltem := oltems:Add("Slider")
    oltem:AllowEdit := 3/*exItemEditSlider*/
    oltem:EditWidth := -128
    oltem:EditValue := 25
oltem1 := oltems:Add("Slider")
    oltem1:AllowEdit := 3/*exItemEditSlider*/
    oltem1:EditBorder := 0/*exEditBorderNone*/
    oltem1:EditWidth := -128
    oltem1:SetProperty("EditOption",1/*exEditMinValue*/,50)
    oltem1:SetProperty("EditOption",2/*exEditMaxValue*/,450)
    oltem1:SetProperty("EditOption",3/*exEditTickStyle*/,2)
    oltem1:SetProperty("EditOption",4/*exEditTickFrequency*/,50)
    oltem1:SetProperty("EditOption",5/*exEditTickLabel*/,"value = vmin ?
' <br> <font ;6> <b>' + value : ( value = vmax ? ' <br> <font ;6> <b>' + value : ( value =
200 ? ' <br> <font ;6> <b> <fgcolor FF0000>' + value : '' ) )")
    oltem1:EditValue := 345
oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

property Item.EditValue as Variant

Specifies the edit's value when the item contains an edit control.

Type	Description
Variant	A VARIANT expression that specifies the edit's value.

The EditValue/EditCaption property specifies the caption to be shown on the item's edit text box. Use the [AllowEdit](#) property to add a text-box inside the item, so the user can type any characters inside. The [EditWidth](#) property specifies the width of the text-box inside the item. The [EditBorder](#) property specifies the border to be shown around the item's text box. You can use the [Get](#) method to collect all items of Edit type. The [EditChange](#) event notifies your application once the user alters the item's text-box caption.

The EditValue property indicates the edit's value as shown bellow:

- The EditValue property specifies the value of the edit field (string expression), without the masking characters, when [AllowEdit](#) property includes the exltemEditMask flag.
- The EditValue property indicates the current slider position/value (long expression), when [AllowEdit](#) property includes the exltemEditSlider, exltemEditProgress, exltemEditScrollBar, or exltemEditColor flag.

How can I assign a color field to the item (method 2)?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("Color")
      .AllowEdit = 6
      .EditBorder = 0
      .EditWidth = -128
      .EditValue = 255
    End With
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
```

```
With .Add("Color")
    .AllowEdit = exItemEditColor
    .EditBorder = exEditBorderNone
    .EditWidth = -128
    .EditValue = 255
End With
End With
.Refresh
End With
```

VB.NET

```
With Exribbon1
    With .Items
        With .Add("Color")
            .AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditColor
            .EditBorder = exontrol.EXRIBBONLib.EditBorderEnum.exEditBorderNone
            .EditWidth = -128
            .EditValue = 255
        End With
    End With
    .Refresh()
End With
```

VB.NET for /COM

```
With AxRibbon1
    With .Items
        With .Add("Color")
            .AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditColor
            .EditBorder = EXRIBBONLib.EditBorderEnum.exEditBorderNone
            .EditWidth = -128
            .EditValue = 255
        End With
    End With
    .Refresh()
End With
```

C++

```
/*  
    Copy and paste the following directives to your header file as  
    it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control  
    Library'  
  
    #import <ExRibbon.dll>  
    using namespace EXRIBBONLib;  
*/  
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-  
>GetControlUnknown();  
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();  
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Color",vtMissing,vtMissing);  
        var_Item->PutAllowEdit(EXRIBBONLib::exItemEditColor);  
        var_Item->PutEditBorder(EXRIBBONLib::exEditBorderNone);  
        var_Item->PutEditWidth(-128);  
        var_Item->PutEditValue(long(255));  
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;  
    Exribbonlib_tlb::IItemPtr var_Item = var_Items-  
>Add(L"Color",TNoParam(),TNoParam());  
        var_Item->AllowEdit = Exribbonlib_tlb::AllowEditEnum::exItemEditColor;  
        var_Item->EditBorder = Exribbonlib_tlb::EditBorderEnum::exEditBorderNone;  
        var_Item->EditWidth = -128;  
        var_Item->set_EditValue(TVariant(255));  
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
    exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Color",null,null);  
        var_Item.AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditColor;
```



```
var_Item.EditBorder = exontrol.EXRIBBONLib.EditBorderEnum.exEditBorderNone;
var_Item.EditWidth = -128;
var_Item.EditValue = 255;
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var var_Item = var_Items.Add("Color",null,null);
    var_Item.AllowEdit = 6;
    var_Item.EditBorder = 0;
    var_Item.EditWidth = -128;
    var_Item.EditValue = 255;
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            With .Add("Color")
```

```

        .AllowEdit = 6
        .EditBorder = 0
        .EditWidth = -128
        .EditValue = 255
    End With
End With
.Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
EXRIBBONLib.Item var_Item = var_Items.Add("Color",null,null);
    var_Item.AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditColor;
    var_Item.EditBorder = EXRIBBONLib.EditBorderEnum.exEditBorderNone;
    var_Item.EditWidth = -128;
    var_Item.EditValue = 255;
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("Color"); com_Item = var_Item;
    com_Item.AllowEdit(6/*exItemEditColor*/);
    com_Item.EditBorder(0/*exEditBorderNone*/);

```

```

com_Item.EditWidth(-128);
com_Item.EditValue(COMVariant::createFromInt(255));
exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
  with Items do
  begin
    with Add('Color',Nil,Nil) do
    begin
      AllowEdit := EXRIBBONLib.AllowEditEnum.exItemEditColor;
      EditBorder := EXRIBBONLib.EditBorderEnum.exEditBorderNone;
      EditWidth := -128;
      EditValue := TObject(255);
    end;
  end;
  Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  with Items do
  begin
    with Add('Color',Null,Null) do
    begin
      AllowEdit := EXRIBBONLib_TLB.exItemEditColor;
      EditBorder := EXRIBBONLib_TLB.exEditBorderNone;
      EditWidth := -128;
      EditValue := OleVariant(255);
    end;
  end;
  Refresh();
end

```

VFP

```
with thisform.Ribbon1
  with .Items
    with .Add("Color")
      .AllowEdit = 6
      .EditBorder = 0
      .EditWidth = -128
      .EditValue = 255
    endwhile
  endwhile
.Refresh
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Items

oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("Color")
var_Item.AllowEdit = 6
var_Item.EditBorder = 0
var_Item.EditWidth = -128
var_Item.EditValue = 255
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
Dim var_Item as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Item = var_Items.Add("Color")
var_Item.AllowEdit = 6
```

```
var_Item.EditBorder = 0
var_Item.EditWidth = -128
var_Item.EditValue = 255
oRibbon.Refresh()
```

Visual Objects

```
local var_Item as IItem
local var_Items as IItems

var_Items := oDCOCX_Exontrol1:Items
var_Item := var_Items:Add("Color",nil,nil)
var_Item:AllowEdit := exItemEditColor
var_Item.EditBorder := exEditBorderNone
var_Item.EditWidth := -128
var_Item:EditValue := 255
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Items

oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Item = var_Items.Add("Color")
var_Item.AllowEdit = 6
var_Item.EditBorder = 0
var_Item.EditWidth = -128
var_Item.EditValue = 255
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
```

```

Variant voltems
Get ComItems to voltems
Handle holtems
Get Create (RefClass(cComItems)) to holtems
Set pvComObject of holtems to voltems
    Variant voltem
    Get ComAdd of holtems "Color" Nothing Nothing to voltem
    Handle holtem
    Get Create (RefClass(cComItem)) to holtem
    Set pvComObject of holtem to voltem
        Set ComAllowEdit of holtem to OLEexItemEditColor
        Set ComEditBorder of holtem to OLEexEditBorderNone
        Set ComEditWidth of holtem to -128
        Set ComEditValue of holtem to 255
    Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( „{100,100}, {640,480}„, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )

```

```
oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-  
CFBE431702E2}*/
```

```
oRibbon:create(, {10,60},{610,370} )
```

```
oItems := oRibbon:Items()
```

```
oItem := oItems:Add("Color")
```

```
oItem:AllowEdit := 6/*exItemEditColor*/
```

```
oItem:EditBorder := 0/*exEditBorderNone*/
```

```
oItem:EditWidth := -128
```

```
oItem:EditValue := 255
```

```
oRibbon:Refresh()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
  nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
  oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Item.EditWidth as Long

Specifies the width for the inside edit control.

Type	Description
Long	A Long expression that specifies the width of the item's text box.

The EditWidth property specifies the width of the text-box inside the item. The [EditBorder](#) property specifies the border to be shown around the item's text box. The [EditCaption](#) property specifies the caption to be shown on the item's Edit text box. Use the [AllowEdit](#) property to add a text-box inside the item, so the user can type any characters inside. You can use the [Get](#) method to collect all items of Edit type. The [EditChange](#) event notifies your application once the user alters the item's text-box caption.

How can I specify the width of the item's editor?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("Mask")
      .AllowEdit = 1
      .EditValue = "this is a text"
      .EditWidth = -128
    End With
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    With .Add("Mask")
      .AllowEdit = exItemEditText
      .EditValue = "this is a text"
      .EditWidth = -128
    End With
  End With
  .Refresh
```


End With

VB.NET

```
With Exribbon1
  With .Items
    With .Add("Mask")
      .AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditText
      .EditValue = "this is a text"
      .EditWidth = -128
    End With
  End With
  .Refresh()
End With
```

VB.NET for /COM

```
With AxRibbon1
  With .Items
    With .Add("Mask")
      .AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditText
      .EditValue = "this is a text"
      .EditWidth = -128
    End With
  End With
  .Refresh()
End With
```

C++

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

  #import <ExRibbon.dll>
  using namespace EXRIBBONLib;
*/
```

```

EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Mask",vtMissing,vtMissing);
        var_Item->PutAllowEdit(EXRIBBONLib::exItemEditText);
        var_Item->PutEditValue("this is a text");
        var_Item->PutEditWidth(-128);
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    Exribbonlib_tlb::IItemPtr var_Item = var_Items-
>Add(L"Mask",TNoParam(),TNoParam());
        var_Item->AllowEdit = Exribbonlib_tlb::AllowEditEnum::exItemEditText;
        var_Item->set_EditValue(TVariant("this is a text"));
        var_Item->EditWidth = -128;
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
    exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Mask",null,null);
        var_Item.AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditText;
        var_Item.EditValue = "this is a text";
        var_Item.EditWidth = -128;
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"></OBJECT>

<SCRIPT LANGUAGE="JScript">

```

```

function Init()
{
    var var_Items = Ribbon1.Items;
    var var_Item = var_Items.Add("Mask",null,null);
    var_Item.AllowEdit = 1;
    var_Item.EditValue = "this is a text";
    var_Item.EditWidth = -128;
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            With .Add("Mask")
                .AllowEdit = 1
                .EditValue = "this is a text"
                .EditWidth = -128
            End With
        End With
    End With
    .Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
EXRIBBONLib.Item var_Item = var_Items.Add("Mask",null,null);
var_Item.AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditText;
var_Item.EditValue = "this is a text";
var_Item.EditWidth = -128;
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("Mask"); com_Item = var_Item;
    com_Item.AllowEdit(1/*exItemEditText*/);
    com_Item.EditValue("this is a text");
    com_Item.EditWidth(-128);
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        with Add('Mask',Nil,Nil) do
        begin
            AllowEdit := EXRIBBONLib.AllowEditEnum.exItemEditText;
            EditValue := 'this is a text';
            EditWidth := -128;
        end;
    end;
end;

```

```
end;  
Refresh();  
end
```

Delphi (standard)

```
with Ribbon1 do  
begin  
  with Items do  
  begin  
    with Add('Mask',Null,Null) do  
    begin  
      AllowEdit := EXRIBBONLib_TLB.exItemEditText;  
      EditValue := 'this is a text';  
      EditWidth := -128;  
    end;  
  end;  
  Refresh();  
end
```

VFP

```
with thisform.Ribbon1  
  with .Items  
    with .Add("Mask")  
      .AllowEdit = 1  
      .EditValue = "this is a text"  
      .EditWidth = -128  
    endwith  
  endwith  
  .Refresh  
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Items  
  
oRibbon = form.Activex1.nativeObject
```

```
var_Items = oRibbon.Items
var_Item = var_Items.Add("Mask")
var_Item.AllowEdit = 1
var_Item.EditValue = "this is a text"
var_Item.EditWidth = -128
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
Dim var_Item as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Item = var_Items.Add("Mask")
var_Item.AllowEdit = 1
var_Item.EditValue = "this is a text"
var_Item.EditWidth = -128
oRibbon.Refresh()
```

Visual Objects

```
local var_Item as IItem
local var_Items as IItems

var_Items := oDCOCX_Exontrol1:Items
var_Item := var_Items:Add("Mask",nil,nil)
var_Item.AllowEdit := exItemEditText
var_Item.EditValue := "this is a text"
var_Item.EditWidth := -128
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Items
```

```
oRibbon = ole_1.Object  
var_Items = oRibbon.Items  
var_Item = var_Items.Add("Mask")  
var_Item.AllowEdit = 1  
var_Item.EditValue = "this is a text"  
var_Item.EditWidth = -128  
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate  
Forward Send OnCreate  
Variant voltems  
Get ComItems to voltems  
Handle holtems  
Get Create (RefClass(cComItems)) to holtems  
Set pvComObject of holtems to voltems  
Variant voltem  
Get ComAdd of holtems "Mask" Nothing Nothing to voltem  
Handle holtem  
Get Create (RefClass(cComItem)) to holtem  
Set pvComObject of holtem to voltem  
Set ComAllowEdit of holtem to OLEexItemEditText  
Set ComEditValue of holtem to "this is a text"  
Set ComEditWidth of holtem to -128  
Send Destroy to holtem  
Send Destroy to holtems  
Send ComRefresh  
End_Procedure
```

XBase++

```
#include "AppEvent.ch"  
#include "ActiveX.ch"
```

PROCEDURE Main

LOCAL oForm

LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

LOCAL oltem

LOCAL oltems

LOCAL oRibbon

oForm := XbpDialog():new(AppDesktop())

oForm:drawingArea:clipChildren := .T.

oForm:create(,, {100,100}, {640,480},,, .F.)

oForm:close := {|| PostAppEvent(xbeP_Quit)}

oRibbon := XbpActiveXControl():new(oForm:drawingArea)

oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/

oRibbon:create(,, {10,60},{610,370})

oltems := oRibbon:Items()

oltem := oltems:Add("Mask")

oltem:**AllowEdit** := 1/*exItemEditText*/

oltem:**EditValue** := "this is a text"

oltem:EditWidth := -128

oRibbon:Refresh()

oForm:Show()

DO WHILE nEvent != xbeP_Quit

nEvent := AppEvent(@mp1, @mp2, @oXbp)

oXbp:handleEvent(nEvent, mp1, mp2)

ENDDO

RETURN

property Item.Enabled as Boolean

Retrieves or sets a value that indicates whether the item is enabled or disabled.

Type	Description
Boolean	A Boolean expression that specifies whether the item is enabled or disabled.

By default, the Enabled property is True. Use the Enabled property to disable an item. A disabled item (Enabled property is False) shows as grayed, and it is un-selectable, so the user can select or highlight it. An item (Enabled property is True, [ShowAsDisabled](#) property is True), shows as grayed, but it is selectable, so the user can select or highlight it. You can use the [Visible](#) property to show or hide the item. The [Remove](#) method removes an individual Item object giving its identifier or caption. Use the [ShowAsDisabled](#) property to show the item as disabled, while it is enabled.

How can I enable or disable an item?

VBA (MS Access, Excell...)

```
With Ribbon1
    With .Items
        .Add("Item").Enabled = False
        .Add("").ToString = "Item[dis]"
    End With
    .Refresh
End With
```

VB6

```
With Ribbon1
    With .Items
        .Add("Item").Enabled = False
        .Add("").ToString = "Item[dis]"
    End With
    .Refresh
End With
```

VB.NET

```
With Exribbon1
```

```

With .Items
    .Add("Item").Enabled = False
    .Add("").ToString = "Item[dis]"
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    With .Items
        .Add("Item").Enabled = False
        .Add("").ToString = "Item[dis]"
    End With
    .Refresh()
End With

```

C++

```

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    var_Items->Add(L"Item",vtMissing,vtMissing)->PutEnabled(VARIANT_FALSE);
    var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"Item[dis]");
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;

```

```
var_Items->Add(L"Item",TNoParam(),TNoParam())->Enabled = false;  
var_Items->Add(L"",TNoParam(),TNoParam())->ToString = L"Item[dis]";  
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
var_Items.Add("Item",null,null).Enabled = false;  
var_Items.Add("",null,null).ToString = "Item[dis]";  
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    var var_Items = Ribbon1.Items;  
    var_Items.Add("Item",null,null).Enabled = false;  
    var_Items.Add("",null,null).ToString = "Item[dis]";  
    Ribbon1.Refresh();  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">
```

```

Function Init()
  With Ribbon1
    With .Items
      .Add("Item").Enabled = False
      .Add("").ToString = "Item[dis]"
    End With
  .Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
  var_Items.Add("Item",null,null).Enabled = false;
  var_Items.Add("",null,null).ToString = "Item[dis]";
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
  COM com_Item,com_Items;
  anytype var_Item,var_Items;
  ;

  super();

  var_Items = exribbon1.Items(); com_Items = var_Items;
  var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
  com_Item.Enabled(false);
  var_Item = COM::createFromObject(com_Items.Add("")); com_Item = var_Item;
  com_Item.ToString("Item[dis]");
  exribbon1.Refresh();

```

```
}
```

Delphi 8 (.NET only)

```
with AxRibbon1 do
begin
  with Items do
  begin
    Add('Item',Nil,Nil).Enabled := False;
    Add('',Nil,Nil).ToString := 'Item[dis]';
  end;
  Refresh();
end
```

Delphi (standard)

```
with Ribbon1 do
begin
  with Items do
  begin
    Add('Item',Null,Null).Enabled := False;
    Add('',Null,Null).ToString := 'Item[dis]';
  end;
  Refresh();
end
```

VFP

```
with thisform.Ribbon1
  with .Items
    .Add("Item").Enabled = .F.
    .Add("").ToString = "Item[dis]"
  endwith
  .Refresh
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Item1,var_Items
```

```

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
// var_Items.Add("Item").Enabled = false
var_Item = var_Items.Add("Item")
with (oRibbon)
    TemplateDef = [Dim var_Item]
    TemplateDef = var_Item
    Template = [var_Item.Enabled = false]
endwith
// var_Items.Add("").ToString = "Item[dis]"
var_Item1 = var_Items.Add("")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.ToString = "Item[dis]"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
' var_Items.Add("Item").Enabled = .f.
var_Item = var_Items.Add("Item")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.Enabled = False"

' var_Items.Add("").ToString = "Item[dis]"
var_Item1 = var_Items.Add("")

```

```
oRibbon.TemplateDef = "Dim var_Item1"  
oRibbon.TemplateDef = var_Item1  
oRibbon.Template = "var_Item1.ToString = \"Item[dis]\""  
  
oRibbon.Refresh()
```

Visual Objects

```
local var_Items as IItems  
  
var_Items := oDCOCX_Exontrol1.Items  
var_Items.Add("Item",nil,nil):Enabled := false  
var_Items.Add("",nil,nil):ToString := "Item[dis]"  
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Items  
  
oRibbon = ole_1.Object  
var_Items = oRibbon.Items  
var_Items.Add("Item").Enabled = false  
var_Items.Add("").ToString = "Item[dis]"  
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate  
Forward Send OnCreate  
Variant voltems  
Get ComItems to voltems  
Handle holtems  
Get Create (RefClass(cComItems)) to holtems  
Set pvComObject of holtems to voltems  
Variant voltem
```

```

Get ComAdd of holtems "Item" Nothing Nothing to voltem
Handle holtem
Get Create (RefClass(cComItem)) to holtem
Set pvComObject of holtem to voltem
    Set ComEnabled of holtem to False
Send Destroy to holtem
Variant voltem1
Get ComAdd of holtems "" Nothing Nothing to voltem1
Handle holtem1
Get Create (RefClass(cComItem)) to holtem1
Set pvComObject of holtem1 to voltem1
    Set ComToString of holtem1 to "Item[dis]"
Send Destroy to holtem1
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

```



```
oItems := oRibbon:Items()  
  oItems:Add("Item"):Enabled := .F.  
  oItems:Add(""):ToString := "Item[dis]"  
oRibbon:Refresh()
```

```
oForm:Show()  
DO WHILE nEvent != xbeP_Quit  
  nEvent := AppEvent( @mp1, @mp2, @oXbp )  
  oXbp:handleEvent( nEvent, mp1, mp2 )  
ENDDO  
RETURN
```

property Item.ForeColor as Color

Specifies the item's foreground color of the item.

Type	Description
Color	A Color expression that specifies the item's foreground color.

The ForeColor property specifies the item's foreground color. The [BackColor](#) property specifies a different background color or a visual appearance for the item. The [Caption](#) property indicates the item's caption to be shown on the item. You can use the <fgcolor> HTML tag in the Caption property to specify a different foreground color for parts of the caption. The [SelfForeColor](#) property specifies the item's foreground color when it is selected or highlighted.

How can I change the item's foreground color?

VBA (MS Access, Excell...)

```
With Ribbon1
    With .Items
        .Add("Item").ForeColor = RGB(255,0,0)
        .Add "<fgcolor FF0000> Item"
        .Add("").ToString = "Item[fg=RGB(255,0,0)]"
    End With
    .Refresh
End With
```

VB6

```
With Ribbon1
    With .Items
        .Add("Item").ForeColor = RGB(255,0,0)
        .Add "<fgcolor FF0000> Item"
        .Add("").ToString = "Item[fg=RGB(255,0,0)]"
    End With
    .Refresh
End With
```

VB.NET

With ExRibbon1

With .Items

.Add("Item").**ForeColor** = Color.FromArgb(255,0,0)

.Add("<fgcolor FF0000> Item")

.Add("").ToString = "Item[fg=RGB(255,0,0)]"

End With

.Refresh()

End With

VB.NET for /COM

With AxRibbon1

With .Items

.Add("Item").**ForeColor** = RGB(255,0,0)

.Add("<fgcolor FF0000> Item")

.Add("").ToString = "Item[fg=RGB(255,0,0)]"

End With

.Refresh()

End With

C++

/*

Copy and paste the following directives to your header file as
it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control
Library'

#import <ExRibbon.dll>

using namespace EXRIBBONLib;

*/

EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();

EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();

var_Items->Add(L"Item",vtMissing,vtMissing)->**PutForeColor**(RGB(255,0,0));

var_Items->Add(L"<fgcolor FF0000> Item",vtMissing,vtMissing);

var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"Item[fg=RGB(255,0,0)]");
spRibbon1->Refresh();

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
var_Items->Add(L"Item",TNoParam(),TNoParam())->ForeColor = RGB(255,0,0);
var_Items->Add(L"<fgcolor FF0000>Item",TNoParam(),TNoParam());
var_Items->Add(L"",TNoParam(),TNoParam())->ToString =
L"Item[fg=RGB(255,0,0)]";
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
var_Items.Add("Item",null,null).ForeColor = Color.FromArgb(255,0,0);
var_Items.Add("<fgcolor FF0000>Item",null,null);
var_Items.Add("",null,null).ToString = "Item[fg=RGB(255,0,0)]";
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var_Items.Add("Item",null,null).ForeColor = 255;
    var_Items.Add("<fgcolor FF0000>Item",null,null);
    var_Items.Add("",null,null).ToString = "Item[fg=RGB(255,0,0)]";
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>
```

VBScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            .Add("Item").ForeColor = RGB(255,0,0)
            .Add "<fgcolor FF0000> Item"
            .Add("").ToString = "Item[fg=RGB(255,0,0)]"
        End With
        .Refresh
    End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.Add("Item",null,null).ForeColor =
(uint)ColorTranslator.ToWin32(Color.FromArgb(255,0,0));
var_Items.Add("<fgcolor FF0000> Item",null,null);
var_Items.Add("",null,null).ToString = "Item[fg=RGB(255,0,0)]";
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

```

```

super();

var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
    com_Item.ForeColor(WinApi::RGB2int(255,0,0));
    com_Items.Add("<fgcolor FF0000>Item");
    var_Item = COM::createFromObject(com_Items.Add("")); com_Item = var_Item;
    com_Item.ToString("Item[fg=RGB(255,0,0)]");
exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        Add('Item',Nil,Nil).ForeColor := $ff;
        Add('<fgcolor FF0000>Item',Nil,Nil);
        Add('',Nil,Nil).ToString := 'Item[fg=RGB(255,0,0)]';
    end;
    Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin
        Add('Item',Null,Null).ForeColor := $ff;
        Add('<fgcolor FF0000>Item',Null,Null);
        Add('',Null,Null).ToString := 'Item[fg=RGB(255,0,0)]';
    end;
    Refresh();
end

```

VFP

```
with thisform.Ribbon1
  with .Items
    .Add("Item").ForeColor = RGB(255,0,0)
    .Add("<fgcolor FF0000>Item")
    .Add("").ToString = "Item[fg=RGB(255,0,0)]"
  endwhile
  .Refresh
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Item1,var_Items

oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
// var_Items.Add("Item").ForeColor = 0xff
var_Item = var_Items.Add("Item")
with (oRibbon)
  TemplateDef = [Dim var_Item]
  TemplateDef = var_Item
  Template = [var_Item.ForeColor = 0xff]
endwith
var_Items.Add("<fgcolor FF0000>Item")
// var_Items.Add("").ToString = "Item[fg=RGB(255,0,0)]"
var_Item1 = var_Items.Add("")
with (oRibbon)
  TemplateDef = [Dim var_Item1]
  TemplateDef = var_Item1
  Template = [var_Item1.ToString = "Item[fg=RGB(255,0,0)]"]
endwith
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
```

```
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P
```

```
oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
```

```
' var_Items.Add("Item").ForeColor = 255
```

```
var_Item = var_Items.Add("Item")
```

```
oRibbon.TemplateDef = "Dim var_Item"
```

```
oRibbon.TemplateDef = var_Item
```

```
oRibbon.Template = "var_Item.ForeColor = 255"
```

```
var_Items.Add("<fgcolor FF0000>Item")
```

```
' var_Items.Add("").ToString = "Item[fg=RGB(255,0,0)]"
```

```
var_Item1 = var_Items.Add("")
```

```
oRibbon.TemplateDef = "Dim var_Item1"
```

```
oRibbon.TemplateDef = var_Item1
```

```
oRibbon.Template = "var_Item1.ToString = \"Item[fg=RGB(255,0,0)]\""
```

```
oRibbon.Refresh()
```

Visual Objects

```
local var_Items as IItems
```

```
var_Items := oDCOCX_Exontrol1.Items
```

```
var_Items.Add("Item",nil,nil):ForeColor := RGB(255,0,0)
```

```
var_Items.Add("<fgcolor FF0000>Item",nil,nil)
```

```
var_Items.Add("",nil,nil):ToString := "Item[fg=RGB(255,0,0)]"
```

```
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Items
```

```
oRibbon = ole_1.Object
```



```

var_Items = oRibbon.Items
var_Items.Add("Item").ForeColor = RGB(255,0,0)
var_Items.Add("<fgcolor FF0000>Item")
var_Items.Add("").ToString = "Item[fg=RGB(255,0,0)]"
oRibbon.Refresh()

```

Visual DataFlex

```

Procedure OnCreate
  Forward Send OnCreate
  Variant voltems
  Get ComItems to voltems
  Handle holtems
  Get Create (RefClass(cComItems)) to holtems
  Set pvComObject of holtems to voltems
  Variant voltem
  Get ComAdd of holtems "Item" Nothing Nothing to voltem
  Handle holtem
  Get Create (RefClass(cComItem)) to holtem
  Set pvComObject of holtem to voltem
  Set ComForeColor of holtem to (RGB(255,0,0))
  Send Destroy to holtem
  Get ComAdd of holtems "<fgcolor FF0000>Item" Nothing Nothing to Nothing
  Variant voltem1
  Get ComAdd of holtems "" Nothing Nothing to voltem1
  Handle holtem1
  Get Create (RefClass(cComItem)) to holtem1
  Set pvComObject of holtem1 to voltem1
  Set ComToString of holtem1 to "Item[fg=RGB(255,0,0)]"
  Send Destroy to holtem1
  Send Destroy to holtems
  Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"

```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
    LOCAL oForm
```

```
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
    LOCAL oltems
```

```
    LOCAL oRibbon
```

```
    oForm := XbpDialog():new( AppDesktop() )
```

```
    oForm:drawingArea:clipChildren := .T.
```

```
    oForm:create( ,, {100,100}, {640,480},,, .F. )
```

```
    oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
```

```
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-  
CFBE431702E2}*/
```

```
    oRibbon:create(,, {10,60},{610,370} )
```

```
        oltems := oRibbon:Items()
```

```
            oltems:Add("Item"):SetProperty("ForeColor",AutomationTranslateColor(  
GraMakeRGBColor ( { 255,0,0 } ) , .F. ))
```

```
            oltems:Add("<fgcolor FF0000> Item")
```

```
            oltems:Add(""):ToString := "Item[fg=RGB(255,0,0)]"
```

```
        oRibbon:Refresh()
```

```
    oForm:Show()
```

```
    DO WHILE nEvent != xbeP_Quit
```

```
        nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
        oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
    ENDDO
```

```
RETURN
```

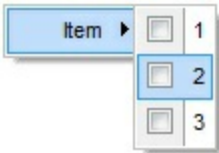
property Item.GroupPopup as GroupPopupEnum

Specifies whether the items of the sub-menu are grouped and displayed on the current item.

Type	Description
GroupPopupEnum	A GroupPopupEnum expression that specifies whether the items of the sub-menu are grouped and displayed on the current item.

By default, the GroupPopup property is exNoGroupPopup, which indicates that the sub-menu is not shown as grouped. Use the GroupPopup property to show the item's sub-menu as a group in the current item. The [SubMenu](#) property indicates the item's sub-menu. The GroupPopup has no effect if the item has no sub-items.

The following screen shot shows the items with no grouping:



The following screen shot shows the items with grouping (horizontally):



The following screen shot shows different type of items grouped horizontally, vertically:



How can I add new items arranged horizontally to the ribbon control (method 2)?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("",2)
      With .Items
        .Add "Item 1"
        .Add "Item 2"
```

```

        .Add "Item 3"
    End With
    .GroupPopup = 3 ' GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
    End With
End With
.Refresh
End With

```

VB6

```

With Ribbon1
    With .Items
        With .Add("",2)
            With .Items
                .Add "Item 1"
                .Add "Item 2"
                .Add "Item 3"
            End With
            .GroupPopup = GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
        End With
    End With
    .Refresh
End With

```

VB.NET

```

With Exribbon1
    With .Items
        With .Add("",2)
            With .Items
                .Add("Item 1")
                .Add("Item 2")
                .Add("Item 3")
            End With
            .GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or

```

```

exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup
    End With
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    With .Items
        With .Add("",2)
            With .Items
                .Add("Item 1")
                .Add("Item 2")
                .Add("Item 3")
            End With
            .GroupPopup = EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or
EXRIBBONLib.GroupPopupEnum.exGroupPopup
        End With
    End With
.Refresh()
End With

```

C++

```

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"",long(2),vtMissing);
EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();

```

```

var_Items1->Add(L"Item 1",vtMissing,vtMissing);
var_Items1->Add(L"Item 2",vtMissing,vtMissing);
var_Items1->Add(L"Item 3",vtMissing,vtMissing);
var_Item-
> PutGroupPopup(EXRIBBONLib::GroupPopupEnum(EXRIBBONLib::exNoGroupPopupFrame | EXRIBBONLib::exGroupPopup));
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
Exribbonlib_tlb::IItemPtr var_Item = var_Items->Add(L"",TVariant(2),TNoParam());
Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;
var_Items1->Add(L"Item 1",TNoParam(),TNoParam());
var_Items1->Add(L"Item 2",TNoParam(),TNoParam());
var_Items1->Add(L"Item 3",TNoParam(),TNoParam());
var_Item->GroupPopup =
Exribbonlib_tlb::GroupPopupEnum::exNoGroupPopupFrame |
Exribbonlib_tlb::GroupPopupEnum::exGroupPopup;
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
var_Items1.Add("Item 1",null,null);
var_Items1.Add("Item 2",null,null);
var_Items1.Add("Item 3",null,null);
var_Item.GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup;
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var var_Item = var_Items.Add("",2,null);
    var var_Items1 = var_Item.Items;
    var_Items1.Add("Item 1",null,null);
    var_Items1.Add("Item 2",null,null);
    var_Items1.Add("Item 3",null,null);
    var_Item.GroupPopup = 3;
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            With .Add("",2)
                With .Items
                    .Add "Item 1"
                    .Add "Item 2"
                    .Add "Item 3"
                End With
            .GroupPopup = 3 ' GroupPopupEnum.exNoGroupPopupFrame Or

```

GroupPopupEnum.exGroupPopup

```
End With
End With
.Refresh
End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;
EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
EXRIBBONLib.Items var_Items1 = var_Item.Items;
var_Items1.Add("Item 1",null,null);
var_Items1.Add("Item 2",null,null);
var_Items1.Add("Item 3",null,null);
var_Item.GroupPopup =
EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
EXRIBBONLib.GroupPopupEnum.exGroupPopup;
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Item,com_Items,com_Items1;
    anytype var_Item,var_Items,var_Items1;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("",COMVariant::createFromInt(2)); com_Item =
var_Item;
    var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
```



```

        com_Items1.Add("Item 1");
        com_Items1.Add("Item 2");
        com_Items1.Add("Item 3");
        com_Item.GroupPopup(3/*exNoGroupPopupFrame | exGroupPopup*/);
exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        with Add('',TObject(2),Nil) do
        begin
            with Items do
            begin
                Add('Item 1',Nil,Nil);
                Add('Item 2',Nil,Nil);
                Add('Item 3',Nil,Nil);
            end;
            GroupPopup :=
Integer(EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib.GroupPopupEnum.exGroupPopup);
        end;
    end;
    Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin
        with Add('',OleVariant(2),Null) do
        begin
            with Items do

```

```

begin
    Add('Item 1',Null,Null);
    Add('Item 2',Null,Null);
    Add('Item 3',Null,Null);
end;
GroupPopup := Integer(EXRIBBONLib_TLB.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib_TLB.exGroupPopup);
end;
end;
Refresh();
end

```

VFP

```

with thisform.Ribbon1
with .Items
with .Add("",2)
with .Items
    .Add("Item 1")
    .Add("Item 2")
    .Add("Item 3")
endwith
.GroupPopup = 3 && GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
endwith
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Items,var_Items1

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Items1 = var_Item.Items
var_Items1.Add("Item 1")

```

```

var_Items1.Add("Item 2")
var_Items1.Add("Item 3")
var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Items as P
Dim var_Items1 as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Items1 = var_Item.Items
var_Items1.Add("Item 1")
var_Items1.Add("Item 2")
var_Items1.Add("Item 3")
var_Item.GroupPopup = 3 'exNoGroupPopupFrame + exGroupPopup
oRibbon.Refresh()

```

Visual Objects

```

local var_Item as IItem
local var_Items,var_Items1 as IItems

var_Items := oDCOCX_Exontrol1:Items
var_Item := var_Items:Add("",2,nil)
var_Items1 := var_Item:Items
var_Items1:Add("Item 1",nil,nil)
var_Items1:Add("Item 2",nil,nil)
var_Items1:Add("Item 3",nil,nil)
var_Item:GroupPopup := exNoGroupPopupFrame | exGroupPopup
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Items,var_Items1
```

```
oRibbon = ole_1.Object
```

```
var_Items = oRibbon.Items
```

```
var_Item = var_Items.Add("",2)
```

```
var_Items1 = var_Item.Items
```

```
var_Items1.Add("Item 1")
```

```
var_Items1.Add("Item 2")
```

```
var_Items1.Add("Item 3")
```

```
var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
```

```
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
```

```
Forward Send OnCreate
```

```
Variant voltems
```

```
Get ComItems to voltems
```

```
Handle holtems
```

```
Get Create (RefClass(cComItems)) to holtems
```

```
Set pvComObject of holtems to voltems
```

```
Variant voltem
```

```
Get ComAdd of holtems "" 2 Nothing to voltem
```

```
Handle holtem
```

```
Get Create (RefClass(cComItem)) to holtem
```

```
Set pvComObject of holtem to voltem
```

```
Variant voltems1
```

```
Get ComItems of holtem to voltems1
```

```
Handle holtems1
```

```
Get Create (RefClass(cComItems)) to holtems1
```

```
Set pvComObject of holtems1 to voltems1
```

```
Get ComAdd of holtems1 "Item 1" Nothing Nothing to Nothing
```

```
Get ComAdd of holtems1 "Item 2" Nothing Nothing to Nothing
```

```
Get ComAdd of holtems1 "Item 3" Nothing Nothing to Nothing
```

```
Send Destroy to holtems1
```

```
Set ComGroupPopup of holtem to (OLEexNoGroupPopupFrame +
OLEexGroupPopup)
Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
LOCAL oForm
LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
LOCAL oltem
LOCAL oltems, oltems1
LOCAL oRibbon

oForm := XbpDialog():new( AppDesktop() )
oForm:drawingArea:clipChildren := .T.
oForm:create( ,, {100,100}, {640,480},,, .F. )
oForm:close := {|| PostAppEvent( xbeP_Quit )}

oRibbon := XbpActiveXControl():new( oForm:drawingArea )
oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
oRibbon:create(,, {10,60},{610,370} )

oltems := oRibbon:Items()
oltem := oltems:Add("",2)
oltems1 := oltem:Items()
oltems1:Add("Item 1")
oltems1:Add("Item 2")
oltems1:Add("Item 3")
oltem:GroupPopup := 3/*exNoGroupPopupFrame+exGroupPopup*/
oRibbon:Refresh()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Item.HotBackColor as Color

Specifies the hot background color of the item (when the cursor hovers the item).

Type	Description
Color	A Color expression that specifies the item's background color, when the cursor hovers it. 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.

The HotBackColor property specifies a different background color or a visual appearance for the item, when the cursor hovers it. The [BackColor](#) property specifies the item's background color of the item. The [SelBackColor](#) property specifies a different background color or a visual appearance for the item, when item is selected / checked. The [Caption](#) property indicates the item's caption to be shown on the item. You can use the <bgcolor> HTML tag in the Caption property to specify a different background color for parts of the caption. The [SelHotBackColor](#) property specifies a different background color or a visual appearance for the item, when item is selected / checked, and the cursor hovers it.

How can I change the item's background/backcolor, when the cursor hovers it (hot)?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    .Add("Item").HotBackColor = RGB(255,0,0)
    .Add("").ToString = "Item[bghot=RGB(255,0,0)]"
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    .Add("Item").HotBackColor = RGB(255,0,0)
    .Add("").ToString = "Item[bghot=RGB(255,0,0)]"
  End With
```

```
.Refresh  
End With
```

VB.NET

```
With Exribbon1  
    With .Items  
        .Add("Item").HotBackColor = Color.FromArgb(255,0,0)  
        .Add("").ToString = "Item[bghot=RGB(255,0,0)]"  
    End With  
    .Refresh()  
End With
```

VB.NET for /COM

```
With AxRibbon1  
    With .Items  
        .Add("Item").HotBackColor = RGB(255,0,0)  
        .Add("").ToString = "Item[bghot=RGB(255,0,0)]"  
    End With  
    .Refresh()  
End With
```

C++

```
/*  
    Copy and paste the following directives to your header file as  
    it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control  
    Library'  
  
    #import <ExRibbon.dll>  
    using namespace EXRIBBONLib;  
*/  
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-  
>GetControlUnknown();  
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();  
var_Items->Add(L"Item",vtMissing,vtMissing)->PutHotBackColor(RGB(255,0,0));  
var_Items->Add(L"",vtMissing,vtMissing)-
```



```
>PutToString(L"Item[bghot=RGB(255,0,0)]");  
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;  
var_Items->Add(L"Item",TNoParam(),TNoParam())->HotBackColor = RGB(255,0,0);  
var_Items->Add(L"",TNoParam(),TNoParam())->ToString =  
L"Item[bghot=RGB(255,0,0)]";  
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
var_Items.Add("Item",null,null).HotBackColor = Color.FromArgb(255,0,0);  
var_Items.Add("",null,null).ToString = "Item[bghot=RGB(255,0,0)]";  
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
var var_Items = Ribbon1.Items;  
var_Items.Add("Item",null,null).HotBackColor = 255;  
var_Items.Add("",null,null).ToString = "Item[bghot=RGB(255,0,0)]";  
Ribbon1.Refresh();  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            .Add("Item").HotBackColor = RGB(255,0,0)
            .Add("").ToString = "Item[bghot=RGB(255,0,0)]"
        End With
        .Refresh
    End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.Add("Item",null,null).HotBackColor =
(uint)ColorTranslator.ToWin32(Color.FromArgb(255,0,0));
var_Items.Add("",null,null).ToString = "Item[bghot=RGB(255,0,0)]";
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

    super();
```

```

var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
    com_Item.HotBackColor(WinApi::RGB2int(255,0,0));
    var_Item = COM::createFromObject(com_Items.Add("")); com_Item = var_Item;
    com_Item.ToString("Item[bghot=RGB(255,0,0)]");
exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        Add('Item',Nil,Nil).HotBackColor := $ff;
        Add('',Nil,Nil).ToString := 'Item[bghot=RGB(255,0,0)]';
    end;
    Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin
        Add('Item',Null,Null).HotBackColor := $ff;
        Add('',Null,Null).ToString := 'Item[bghot=RGB(255,0,0)]';
    end;
    Refresh();
end

```

VFP

```

with thisform.Ribbon1
    with .Items

```

```

.Add("Item").HotBackColor = RGB(255,0,0)
.Add("").ToString = "Item[bghot=RGB(255,0,0)]"
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Items

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
// var_Items.Add("Item").HotBackColor = 0xff
var_Item = var_Items.Add("Item")
with (oRibbon)
    TemplateDef = [Dim var_Item]
    TemplateDef = var_Item
    Template = [var_Item.HotBackColor = 0xff]
endwith
// var_Items.Add("").ToString = "Item[bghot=RGB(255,0,0)]"
var_Item1 = var_Items.Add("")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.ToString = "Item[bghot=RGB(255,0,0)]"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items =oRibbon.Items

```

```
' var_Items.Add("Item").HotBackColor = 255
```

```
var_Item = var_Items.Add("Item")
```

```
oRibbon.TemplateDef = "Dim var_Item"
```

```
oRibbon.TemplateDef = var_Item
```

```
oRibbon.Template = "var_Item.HotBackColor = 255"
```

```
' var_Items.Add("").ToString = "Item[bghot=RGB(255,0,0)]"
```

```
var_Item1 = var_Items.Add("")
```

```
oRibbon.TemplateDef = "Dim var_Item1"
```

```
oRibbon.TemplateDef = var_Item1
```

```
oRibbon.Template = "var_Item1.ToString = \"Item[bghot=RGB(255,0,0)]\""
```

```
oRibbon.Refresh()
```

Visual Objects

```
local var_Items as IItems
```

```
var_Items := oDCOCX_Exontrol1.Items
```

```
var_Items.Add("Item",nil,nil):HotBackColor := RGB(255,0,0)
```

```
var_Items.Add("",nil,nil):ToString := "Item[bghot=RGB(255,0,0)]"
```

```
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Items
```

```
oRibbon = ole_1.Object
```

```
var_Items = oRibbon.Items
```

```
var_Items.Add("Item").HotBackColor = RGB(255,0,0)
```

```
var_Items.Add("").ToString = "Item[bghot=RGB(255,0,0)]"
```

```
oRibbon.Refresh()
```

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Variant voltems

Get Comltems to voltems

Handle holtems

Get Create (RefClass(cComltems)) to holtems

Set pvComObject of holtems to voltems

Variant voltem

Get ComAdd of holtems "Item" Nothing Nothing to voltem

Handle holtem

Get Create (RefClass(cComltem)) to holtem

Set pvComObject of holtem to voltem

Set **ComHotBackColor** of holtem to (RGB(255,0,0))

Send Destroy to holtem

Variant voltem1

Get ComAdd of holtems "" Nothing Nothing to voltem1

Handle holtem1

Get Create (RefClass(cComltem)) to holtem1

Set pvComObject of holtem1 to voltem1

Set ComToString of holtem1 to "Item[bghot=RGB(255,0,0)]"

Send Destroy to holtem1

Send Destroy to holtems

Send ComRefresh

End_Procedure

XBase++

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
LOCAL oForm
```

```
LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
LOCAL oltems
```

```
LOCAL oRibbon
```

```
oForm := XbpDialog():new( AppDesktop() )
```

```

oForm:drawingArea:clipChildren := .T.
oForm:create( ,, {100,100}, {640,480} ,, .F. )
oForm:close := {|| PostAppEvent( xbeP_Quit )}

oRibbon := XbpActiveXControl():new( oForm:drawingArea )
oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
oRibbon:create(,, {10,60},{610,370} )

    oItems := oRibbon:Items()
        oItems:Add("Item"):SetProperty("HotBackColor",AutomationTranslateColor(
GraMakeRGBColor ( { 255,0,0 } ) , .F. ))
        oItems:Add(""):ToString := "Item[bghot=RGB(255,0,0)]"
oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

property Item.HTMLImage as String

Retrieves or sets a value that indicates the key of the image (HTMLPicture method) to be displayed on the item (left side).

Type	Description
String	A String expression that indicates the key of the picture to be displayed on the left side of the caption.

The HTMLImage property assigns a picture to the left side of the caption. The key of the picture to be displayed must be loaded previously using the [HTMLPicture](#) property. The HTMLImage property has effect only if the Image property is -1 (by default). Use the [Image](#) property to assign an icon from the [Images](#) collection to the left side of the caption. Use the [PopupFlatImageWidth](#) property to specify the width of the column that displays icons/images/check or radio buttons.

The following VFP samples loads the picture using the HTMLPicture method, and displays it on the left side of the caption using the HTMLImage property.

```
ribbon = CreateObject("Exontrol.Ribbon")
with ribbon
.HTMLPicture("pic1") = "C:\exontrol\images\colorize.gif"
.Items.ToString = "Item A[himg=pic1]"
endwith
```

or:

```
ribbon = CreateObject("Exontrol.Ribbon")
with ribbon
.HTMLPicture("pic1") = "C:\exontrol\images\colorize.gif"
.Items.Add("Item A").HTMLImage = "pic1"
endwith
```

These two samples are equivalent.

property Item.ID as Long

Retrieves or sets a value that specifies the item's identifier.

Type	Description
Long	A Long expression that defines the item's identifier. This property can be specified at adding time, by using the ID parameter of the Add method.

Use the ID property to associate an unique identifier to each item. You can use the [Item](#) property of the Ribbon component to get the [Item](#) object based on its identifier. Use the [Debug](#) property to display the identifiers for all visible items, for debugging purposes. The First number in the [] parenthesis indicates the item's ID property. Use the [Caption](#) property to specify the item's caption. Use the [UserData](#) property to associate any extra data to your items.

How can I specify/assign the item's identifier?

VBA (MS Access, Excell...)

```
With Ribbon1
    .Debug = True
    With .Items
        .Add "ID 1",0,1000
        .Add("ID 2").ID = 1001
        .Add("").ToString = "ID 3[id=1002]"
    End With
    .Refresh
End With
```

VB6

```
With Ribbon1
    .Debug = True
    With .Items
        .Add "ID 1",0,1000
        .Add("ID 2").ID = 1001
        .Add("").ToString = "ID 3[id=1002]"
    End With
    .Refresh
End With
```

VB.NET

```
With Exribbon1
    .Debug = True
    With .Items
        .Add("ID 1",0,1000)
        .Add("ID 2").ID = 1001
        .Add("").ToString = "ID 3[id=1002]"
    End With
    .Refresh()
End With
```

VB.NET for /COM

```
With AxRibbon1
    .Debug = True
    With .Items
        .Add("ID 1",0,1000)
        .Add("ID 2").ID = 1001
        .Add("").ToString = "ID 3[id=1002]"
    End With
    .Refresh()
End With
```

C++

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
spRibbon1->PutDebug(VARIANT_TRUE);
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
```

```

var_Items->Add(L"ID 1",long(0),long(1000));
var_Items->Add(L"ID 2",vtMissing,vtMissing)->PutID(1001);
var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"ID 3[id=1002]");
spRibbon1->Refresh();

```

C++ Builder

```

Ribbon1->Debug = true;
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
var_Items->Add(L"ID 1",TVariant(0),TVariant(1000));
var_Items->Add(L"ID 2",TNoParam(),TNoParam())->ID = 1001;
var_Items->Add(L"",TNoParam(),TNoParam())->ToString = L"ID 3[id=1002]";
Ribbon1->Refresh();

```

C#

```

exribbon1.Debug = true;
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
var_Items.Add("ID 1",0,1000);
var_Items.Add("ID 2",null,null).ID = 1001;
var_Items.Add("",null,null).ToString = "ID 3[id=1002]";
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Ribbon1.Debug = true;
    var var_Items = Ribbon1.Items;
    var_Items.Add("ID 1",0,1000);

```

```

var_Items.Add("ID 2",null,null).ID = 1001;
var_Items.Add("",null,null).ToString = "ID 3[id=1002]";
Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        .Debug = True
        With .Items
            .Add "ID 1",0,1000
            .Add("ID 2").ID = 1001
            .Add("").ToString = "ID 3[id=1002]"
        End With
        .Refresh
    End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

axRibbon1.Debug = true;
EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.Add("ID 1",0,1000);
var_Items.Add("ID 2",null,null).ID = 1001;
var_Items.Add("",null,null).ToString = "ID 3[id=1002]";
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

    super();

    exribbon1.Debug(true);
    var_Items = exribbon1.Items(); com_Items = var_Items;
    com_Items.Add("ID
1",COMVariant::createFromInt(0),COMVariant::createFromInt(1000));
    var_Item = COM::createFromObject(com_Items.Add("ID 2")); com_Item =
var_Item;
    com_Item.ID(1001);
    var_Item = COM::createFromObject(com_Items.Add("")); com_Item = var_Item;
    com_Item.ToString("ID 3[id=1002]");
    exribbon1.Refresh();
}
```

Delphi 8 (.NET only)

```
with AxRibbon1 do
begin
    Debug := True;
    with Items do
    begin
        Add('ID 1',TObject(0),TObject(1000));
        Add('ID 2',Nil,Nil).ID := 1001;
        Add('',Nil,Nil).ToString := 'ID 3[id=1002]';
    end;
    Refresh();
end
```

Delphi (standard)

```
with Ribbon1 do
begin
  Debug := True;
  with Items do
  begin
    Add('ID 1',OleVariant(0),OleVariant(1000));
    Add('ID 2',Null,Null).ID := 1001;
    Add('',Null,Null).ToString := 'ID 3[id=1002]';
  end;
  Refresh();
end
```

VFP

```
with thisform.Ribbon1
.Debug = .T.
with .Items
.Add("ID 1",0,1000)
.Add("ID 2").ID = 1001
.Add("").ToString = "ID 3[id=1002]"
endwith
.Refresh
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Item1,var_Items

oRibbon = form.Activex1.nativeObject
oRibbon.Debug = true
var_Items = oRibbon.Items
var_Items.Add("ID 1",0,1000)
// var_Items.Add("ID 2").ID = 1001
var_Item = var_Items.Add("ID 2")
with (oRibbon)
  TemplateDef = [Dim var_Item]
```

```

    TemplateDef = var_Item
    Template = [var_Item.ID = 1001]
endwith
// var_Items.Add("").ToString = "ID 3[id=1002]"
var_Item1 = var_Items.Add("")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.ToString = "ID 3[id=1002]"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
oRibbon.Debug = .t.
var_Items = oRibbon.Items
    var_Items.Add("ID 1",0,1000)
    ' var_Items.Add("ID 2").ID = 1001
    var_Item = var_Items.Add("ID 2")
    oRibbon.TemplateDef = "Dim var_Item"
    oRibbon.TemplateDef = var_Item
    oRibbon.Template = "var_Item.ID = 1001"

    ' var_Items.Add("").ToString = "ID 3[id=1002]"
    var_Item1 = var_Items.Add("")
    oRibbon.TemplateDef = "Dim var_Item1"
    oRibbon.TemplateDef = var_Item1
    oRibbon.Template = "var_Item1.ToString = \"ID 3[id=1002]\" "
oRibbon.Refresh()

```

Visual Objects

```
local var_Items as Items

oDCOCX_Exontrol1:Debug := true
var_Items := oDCOCX_Exontrol1:Items
  var_Items:Add("ID 1",0,1000)
  var_Items:Add("ID 2",nil,nil):ID := 1001
  var_Items:Add("",nil,nil):ToString := "ID 3[id=1002]"
oDCOCX_Exontrol1:Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Items

oRibbon = ole_1.Object
oRibbon.Debug = true
var_Items = oRibbon.Items
  var_Items.Add("ID 1",0,1000)
  var_Items.Add("ID 2").ID = 1001
  var_Items.Add("").ToString = "ID 3[id=1002]"
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
  Forward Send OnCreate
  Set ComDebug to True
  Variant voltems
  Get ComItems to voltems
  Handle holtems
  Get Create (RefClass(cComItems)) to holtems
  Set pvComObject of holtems to voltems
  Get ComAdd of holtems "ID 1" 0 1000 to Nothing
```



```

Variant voltem
Get ComAdd of holtems "ID 2" Nothing Nothing to voltem
Handle holtem
Get Create (RefClass(cComItem)) to holtem
Set pvComObject of holtem to voltem
    Set ComID of holtem to 1001
Send Destroy to holtem
Variant voltem1
Get ComAdd of holtems "" Nothing Nothing to voltem1
Handle holtem1
Get Create (RefClass(cComItem)) to holtem1
Set pvComObject of holtem1 to voltem1
    Set ComToString of holtem1 to "ID 3[id=1002]"
Send Destroy to holtem1
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/

```

```
oRibbon:create(,, {10,60},{610,370} )
```

```
oRibbon:Debug := .T.
```

```
oItems := oRibbon:Items()
```

```
oItems:Add("ID 1",0,1000)
```

```
oItems:Add("ID 2"):ID := 1001
```

```
oItems:Add(""):ToString := "ID 3[id=1002]"
```

```
oRibbon:Refresh()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Item.Image as Long

Retrieves or sets a value that indicates the item's index image.

Type	Description
Long	A Long expression that specifies the zero-based index of the icon to be displayed on the item.

By default, the Image property is -1, which indicates no icon associated. The Image is displayed on the left side of the item's caption. The [Images](#) method should be used to specify a collection of icons to be used by the control. Use the [Replacelcon](#) method to add, remove or clear icons in the control's images collection. The tag can be used in the [Caption](#) property of the [Item](#) object to display an Icon or a custom-size picture. Use the [HTMLImage](#) property to assign a BMP, JPG, GIF or PNG file to left side of the caption, the same way as you will do with the Image property.

How can I can I assign icons for the item?

VBA (MS Access, Excell...)

With Ribbon1

.Images

"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrkltl0vr
& _

"/oFBoVDolFo1HpFJpVLplNp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
& _

"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
& _

"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB
& _

"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA="

With .Items

.Add("Item").Image = 1

.Add("Item").Image = 2

.Add "",1

.Add "Item 1 "

.Add "Item 2 "

.Add "",1

.Add("Item 1 ").Image = 1

End With

.Refresh
End With

VB6

With Ribbon1

.Images

"gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vr
& _
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
& _
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
& _
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB
& _
"NAOAEAwCjMBwFAEDwJBMDwLBYP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA="

With .Items

.Add("Item").Image = 1

.Add("Item").Image = 2

.Add "",1

.Add "Item 1 "

.Add "Item 2 "

.Add "",1

.Add("Item 1 ").Image = 1

End With

.Refresh

End With

VB.NET

With Exribbon1

**.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vr
& _
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
& _
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
& _**

```

"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vI8j4f/qfEZeB
& _
"NAOAEAwCjMBwFAEDwJBMDwLBYP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA=")
With .Items
    .Add("Item").Image = 1
    .Add("Item").Image = 2
    .Add("",1)
    .Add("Item <img>1 </img>")
    .Add("Item <img>2 </img>")
    .Add("",1)
    .Add("Item <img>1 </img>").Image = 1
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1

.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEalEaEEaAlAkcbk0ol
& _
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
& _
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
& _
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vI8j4f/qfEZeB
& _
"NAOAEAwCjMBwFAEDwJBMDwLBYP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA=")
With .Items
    .Add("Item").Image = 1
    .Add("Item").Image = 2
    .Add("",1)
    .Add("Item <img>1 </img>")
    .Add("Item <img>2 </img>")
    .Add("",1)
    .Add("Item <img>1 </img>").Image = 1
End With

```

.Refresh()
End With

C++

```
/*  
    Copy and paste the following directives to your header file as  
    it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control  
    Library'  
  
    #import <ExRibbon.dll>  
    using namespace EXRIBBONLib;  
*/  
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-  
>GetControlUnknown();  
spRibbon1->  
> Images(_bstr_t("gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAl/  
+  
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl/  
+  
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m/  
+  
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB/  
+  
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oyglA="));  
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();  
var_Items->Add(L"Item",vtMissing,vtMissing)->PutImage(1);  
var_Items->Add(L"Item",vtMissing,vtMissing)->PutImage(2);  
var_Items->Add(L"",long(1),vtMissing);  
var_Items->Add(L"Item <img> 1 </img>",vtMissing,vtMissing);  
var_Items->Add(L"Item <img> 2 </img>",vtMissing,vtMissing);  
var_Items->Add(L"",long(1),vtMissing);  
var_Items->Add(L"Item <img> 1 </img>",vtMissing,vtMissing)->PutImage(1);  
spRibbon1->Refresh();
```

C++ Builder

Ribbon1-

```
> Images(TVariant(String("gBJJgBAIDAAGAAEAQAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaI  
+  
"/oFBoVDoIFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl  
+  
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m  
+  
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB  
+  
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA="));
```

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
```

```
var_Items->Add(L"Item",TNoParam(),TNoParam())->Image = 1;
```

```
var_Items->Add(L"Item",TNoParam(),TNoParam())->Image = 2;
```

```
var_Items->Add(L"",TVariant(1),TNoParam());
```

```
var_Items->Add(L"Item <img>1</img>",TNoParam(),TNoParam());
```

```
var_Items->Add(L"Item <img>2</img>",TNoParam(),TNoParam());
```

```
var_Items->Add(L"",TVariant(1),TNoParam());
```

```
var_Items->Add(L"Item <img>1</img>",TNoParam(),TNoParam())->Image = 1;
```

```
Ribbon1->Refresh();
```

C#

```
exribbon1.Images("gBJJgBAIDAAGAAEAQAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaA/  
+  
"/oFBoVDoIFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl  
+  
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m  
+  
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB  
+  
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA=");
```

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
```

```
var_Items.Add("Item",null,null).Image = 1;
```

```
var_Items.Add("Item",null,null).Image = 2;
```

```
var_Items.Add("",1,null);
```

```
var_Items.Add("Item <img>1</img>",null,null);
```

```

var_Items.Add("Item <img>2</img>",null,null);
var_Items.Add("",1,null);
var_Items.Add("Item <img>1</img>",null,null).Image = 1;
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
Ribbon1.Images("gBJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAl,
+
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
+
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
+
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB
+
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oYgIA="");
var var_Items = Ribbon1.Items;
var_Items.Add("Item",null,null).Image = 1;
var_Items.Add("Item",null,null).Image = 2;
var_Items.Add("",1,null);
var_Items.Add("Item <img>1</img>",null,null);
var_Items.Add("Item <img>2</img>",null,null);
var_Items.Add("",1,null);
var_Items.Add("Item <img>1</img>",null,null).Image = 1;
Ribbon1.Refresh();

```



```
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
  With Ribbon1  
    .Images  
    "gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrkltl0vr  
& _  
    "/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl  
& _  
    "/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m  
& _  
    "x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB  
& _  
    "NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oYglA="  
    With .Items  
      .Add("Item").Image = 1  
      .Add("Item").Image = 2  
      .Add "",1  
      .Add "Item <img>1 </img> "  
      .Add "Item <img>2 </img> "  
      .Add "",1  
      .Add("Item <img>1 </img> ").Image = 1  
    End With  
    .Refresh
```

```
End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
axRibbon1.Images("gBJJgBAIDAAGAAEAQAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEa,
+
"/oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxl
+
"/wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m
+
"x3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB
+
"NAOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA=");
EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.Add("Item",null,null).Image = 1;
var_Items.Add("Item",null,null).Image = 2;
var_Items.Add("",1,null);
var_Items.Add("Item <img> 1 </img>",null,null);
var_Items.Add("Item <img> 2 </img>",null,null);
var_Items.Add("",1,null);
var_Items.Add("Item <img> 1 </img>",null,null).Image = 1;
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    str var_s;
    ;

    super();
```

```

var_s =
"gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vr

var_s = var_s +
"oFBoVDolFo1HpFJpVLplNp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxu'

var_s = var_s +
"wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+mC

var_s = var_s +
"3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeBC

var_s = var_s +
"AOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oygIA=";
exribbon1.Images(COMVariant::createFromStr(var_s));
var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
    com_Item.Image(1);
    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
    com_Item.Image(2);
    com_Items.Add("",COMVariant::createFromInt(1));
    com_Items.Add("Item <img> 1 </img>");
    com_Items.Add("Item <img> 2 </img>");
    com_Items.Add("",COMVariant::createFromInt(1));
    var_Item = COM::createFromObject(com_Items.Add("Item <img> 1 </img>"));
com_Item = var_Item;
    com_Item.Image(1);
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin

```

```

Images('gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oII
+
'oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxu\
+
'wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m0
+
'3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEIHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB0
+
'AOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oyglA=');
with Items do
begin
  Add('Item',Nil,Nil).Image := 1;
  Add('Item',Nil,Nil).Image := 2;
  Add('',TObject(1),Nil);
  Add('Item <img>1 </img>',Nil,Nil);
  Add('Item <img>2 </img>',Nil,Nil);
  Add('',TObject(1),Nil);
  Add('Item <img>1 </img>',Nil,Nil).Image := 1;
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin

Images('gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oII
+
'oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxu\
+

```

```

'wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m0
+
'3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB0
+
'AOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oyglA='');
with Items do
begin
  Add('Item',Null,Null).Image := 1;
  Add('Item',Null,Null).Image := 2;
  Add('',OleVariant(1),Null);
  Add('Item <img>1</img>',Null,Null);
  Add('Item <img>2</img>',Null,Null);
  Add('',OleVariant(1),Null);
  Add('Item <img>1</img>',Null,Null).Image := 1;
end;
Refresh();
end

```

VFP

```

with thisform.Ribbon1
  var_s =
'gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vr

  var_s = var_s +
'oFBoVDolFo1HpFJpVLpINp1PqFRqVTqIVq1XrFZrVbrldr1fsFhsVjslls1ntFptVrtltt1vuFxu'

  var_s = var_s +
'wGBwWDwmFw2HxGJxWLxmNx0xiFdyOT8Tf9ZymXx+QytcyNgz8r0ObIWjyWds+m0

  var_s = var_s +
'3GO4NV3WeyvD2XJ5XL5nN51aiw+lfSj0gkUkAEllHanHI5j/cHg8EZf7w8vl8j4f/qfEZeB0

  var_s = var_s +
'AOAEAwCjMBwFAEDwJBMDwLBYAP2/8Hv8/gAGAD8LQs9w/nhDY/oyglA="
.Images(var_s)

```

```

with .Items
    .Add("Item").Image = 1
    .Add("Item").Image = 2
    .Add("",1)
    .Add("Item <img> 1 </img> ")
    .Add("Item <img> 2 </img> ")
    .Add("",1)
    .Add("Item <img> 1 </img> ").Image = 1
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Item2,var_Items

oRibbon = form.ActiveX1.nativeObject
oRibbon.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAl

var_Items = oRibbon.Items
// var_Items.Add("Item").Image = 1
var_Item = var_Items.Add("Item")
with (oRibbon)
    TemplateDef = [Dim var_Item]
    TemplateDef = var_Item
    Template = [var_Item.Image = 1]
endwith
// var_Items.Add("Item").Image = 2
var_Item1 = var_Items.Add("Item")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.Image = 2]
endwith
var_Items.Add("",1)
var_Items.Add("Item <img> 1 </img> ")
var_Items.Add("Item <img> 2 </img> ")

```

```

var_Items.Add("",1)
// var_Items.Add("Item <img>1</img>").Image = 1
var_Item2 = var_Items.Add("Item <img>1</img>")
with (oRibbon)
    TemplateDef = [Dim var_Item2]
    TemplateDef = var_Item2
    Template = [var_Item2.Image = 1]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Item2 as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
oRibbon.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAl

var_Items = oRibbon.Items
' var_Items.Add("Item").Image = 1
var_Item = var_Items.Add("Item")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.Image = 1"

' var_Items.Add("Item").Image = 2
var_Item1 = var_Items.Add("Item")
oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1
oRibbon.Template = "var_Item1.Image = 2"

var_Items.Add("",1)
var_Items.Add("Item <img>1</img>")

```

```

var_Items.Add("Item <img>2</img>")
var_Items.Add("",1)
' var_Items.Add("Item <img>1</img>").Image = 1
var_Item2 = var_Items.Add("Item <img>1</img>")
oRibbon.TemplateDef = "Dim var_Item2"
oRibbon.TemplateDef = var_Item2
oRibbon.Template = "var_Item2.Image = 1"

oRibbon.Refresh()

```

Visual Objects

```

local var_Items as Items

oDCOCX_Exontrol1:Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAl")

var_Items := oDCOCX_Exontrol1:Items
var_Items.Add("Item",nil,nil):Image := 1
var_Items.Add("Item",nil,nil):Image := 2
var_Items.Add("",1,nil)
var_Items.Add("Item <img>1</img>",nil,nil)
var_Items.Add("Item <img>2</img>",nil,nil)
var_Items.Add("",1,nil)
var_Items.Add("Item <img>1</img>",nil,nil):Image := 1
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```

OleObject oRibbon,var_Items

oRibbon = ole_1.Object
oRibbon.Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAl")

var_Items = oRibbon.Items
var_Items.Add("Item").Image = 1
var_Items.Add("Item").Image = 2

```



```

var_Items.Add("",1)
var_Items.Add("Item <img>1</img>")
var_Items.Add("Item <img>2</img>")
var_Items.Add("",1)
var_Items.Add("Item <img>1</img>").Image = 1
oRibbon.Refresh()

```

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Send **ComImages**

"gBJJgBAIDAAGAAEAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAIAkcbk0oIUrlktl0vr

Variant voltems

Get ComItems to voltems

Handle holtems

Get Create (RefClass(cComItems)) to holtems

Set pvComObject of holtems to voltems

Variant voltem

Get **ComAdd** of holtems "Item" Nothing Nothing to voltem

Handle holtem

Get Create (RefClass(cComItem)) to holtem

Set pvComObject of holtem to voltem

Set **ComImage** of holtem to 1

Send Destroy to holtem

Variant voltem1

Get **ComAdd** of holtems "Item" Nothing Nothing to voltem1

Handle holtem1

Get Create (RefClass(cComItem)) to holtem1

Set pvComObject of holtem1 to voltem1

Set **ComImage** of holtem1 to 2

Send Destroy to holtem1

Get **ComAdd** of holtems "" 1 Nothing to Nothing

Get **ComAdd** of holtems "Item 1" Nothing Nothing to Nothing

Get **ComAdd** of holtems "Item 2" Nothing Nothing to Nothing

```

Get ComAdd of holtems "" 1 Nothing to Nothing
Variant voltem2
Get ComAdd of holtems "Item <img>1</img>" Nothing Nothing to voltem2
Handle holtem2
Get Create (RefClass(cComItem)) to holtem2
Set pvComObject of holtem2 to voltem2
    Set ComImage of holtem2 to 1
Send Destroy to holtem2
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oRibbon:Images("gBJJgBAIDAAGAAEAAQhYAf8Pf4hh0QihCJo2AEZjQAjEZFEaIEaEEaAl

    oltems := oRibbon:Items()

```

```
oItems:Add("Item"):Image := 1
oItems:Add("Item"):Image := 2
oItems:Add("",1)
oItems:Add("Item <img>1 </img>")
oItems:Add("Item <img>2 </img>")
oItems:Add("",1)
oItems:Add("Item <img>1 </img>"):Image := 1
oRibbon:Refresh()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Item.Italic as Boolean

Specifies whether the item's caption should appear in italic.

Type	Description
Boolean	A Boolean expression that specifies whether the item's caption is shown in italic.

By default, the Italic property is False. Use the Italic property to show the item's caption in italic. The [Caption](#) property indicates the HTML caption to be shown on the item. The <i> HTML tag can be used on the item's Caption property to specify different parts of the caption to be shown in italic.

How can I show the item in italics?

VBA (MS Access, Excell...)

```
With Ribbon1
    With .Items
        .Add("Item").Italic = True
        .Add "<i>Item</i> "
        .Add("").ToString = "Item[itl]"
    End With
    .Refresh
End With
```

VB6

```
With Ribbon1
    With .Items
        .Add("Item").Italic = True
        .Add "<i>Item</i> "
        .Add("").ToString = "Item[itl]"
    End With
    .Refresh
End With
```

VB.NET

```
With Exribbon1
    With .Items
```

```

.Add("Item").Italic = True
.Add("<i>Item</i>")
.Add("").ToString = "Item[itl]"
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
  With .Items
    .Add("Item").Italic = True
    .Add("<i>Item</i>")
    .Add("").ToString = "Item[itl]"
  End With
  .Refresh()
End With

```

C++

```

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

  #import <ExRibbon.dll>
  using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
var_Items->Add(L"Item",vtMissing,vtMissing)->PutItalic(VARIANT_TRUE);
var_Items->Add(L"<i>Item</i>",vtMissing,vtMissing);
var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"Item[itl]");
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
var_Items->Add(L"Item",TNoParam(),TNoParam())->Italic = true;
var_Items->Add(L"<i>Item</i>",TNoParam(),TNoParam());
var_Items->Add(L"",TNoParam(),TNoParam())->ToString = L"Item[itl]";
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
var_Items.Add("Item",null,null).Italic = true;
var_Items.Add("<i>Item</i>",null,null);
var_Items.Add("",null,null).ToString = "Item[itl]";
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var_Items.Add("Item",null,null).Italic = true;
    var_Items.Add("<i>Item</i>",null,null);
    var_Items.Add("",null,null).ToString = "Item[itl]";
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>

```

```

<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Ribbon1
    With .Items
      .Add("Item").Italic = True
      .Add "<i>Item</i> "
      .Add("").ToString = "Item[itl]"
    End With
    .Refresh
  End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.Add("Item",null,null).Italic = true;
var_Items.Add("<i>Item</i> ",null,null);
var_Items.Add("",null,null).ToString = "Item[itl]";
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
  COM com_Item,com_Items;
  anytype var_Item,var_Items;
  ;

  super();

  var_Items = exribbon1.Items(); com_Items = var_Items;

```

```

    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
    com_Item.Italic(true);
    com_Items.Add("<i>Item</i>");
    var_Item = COM::createFromObject(com_Items.Add("")); com_Item = var_Item;
    com_Item.ToString("Item[itl]");
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
  with Items do
  begin
    Add('Item',Nil,Nil).Italic := True;
    Add('<i>Item</i>',Nil,Nil);
    Add('',Nil,Nil).ToString := 'Item[itl]';
  end;
  Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  with Items do
  begin
    Add('Item',Null,Null).Italic := True;
    Add('<i>Item</i>',Null,Null);
    Add('',Null,Null).ToString := 'Item[itl]';
  end;
  Refresh();
end

```

VFP

```

with thisform.Ribbon1

```



```

with .Items
    .Add("Item").Italic = .T.
    .Add("<i>Item</i> ")
    .Add("").ToString = "Item[itl]"
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Items

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
// var_Items.Add("Item").Italic = true
var_Item = var_Items.Add("Item")
with (oRibbon)
    TemplateDef = [Dim var_Item]
    TemplateDef = var_Item
    Template = [var_Item.Italic = true]
endwith
var_Items.Add("<i>Item</i> ")
// var_Items.Add("").ToString = "Item[itl]"
var_Item1 = var_Items.Add("")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.ToString = "Item[itl]"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P

```

```

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
' var_Items.Add("Item").Italic = .t.
var_Item = var_Items.Add("Item")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.Italic = True"

var_Items.Add("<i>Item</i>")
' var_Items.Add("").ToString = "Item[itl]"
var_Item1 = var_Items.Add("")
oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1
oRibbon.Template = "var_Item1.ToString = \"Item[itl]\""

oRibbon.Refresh()

```

Visual Objects

```

local var_Items as Items

var_Items := oDCOCX_Exontrol1.Items
var_Items.Add("Item",nil,nil):Italic := true
var_Items.Add("<i>Item</i>",nil,nil)
var_Items.Add("",nil,nil):ToString := "Item[itl]"
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```

OleObject oRibbon,var_Items

oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Items.Add("Item").Italic = true
var_Items.Add("<i>Item</i>")

```

```
var_Items.Add("").ToString = "Item[itl]"  
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate  
  Forward Send OnCreate  
  Variant voltems  
  Get ComItems to voltems  
  Handle holtems  
  Get Create (RefClass(cComItems)) to holtems  
  Set pvComObject of holtems to voltems  
    Variant voltem  
    Get ComAdd of holtems "Item" Nothing Nothing to voltem  
    Handle holtem  
    Get Create (RefClass(cComItem)) to holtem  
    Set pvComObject of holtem to voltem  
      Set ComItalic of holtem to True  
    Send Destroy to holtem  
    Get ComAdd of holtems "<i>Item</i>" Nothing Nothing to Nothing  
    Variant voltem1  
    Get ComAdd of holtems "" Nothing Nothing to voltem1  
    Handle holtem1  
    Get Create (RefClass(cComItem)) to holtem1  
    Set pvComObject of holtem1 to voltem1  
      Set ComToString of holtem1 to "Item[itl]"  
    Send Destroy to holtem1  
  Send Destroy to holtems  
  Send ComRefresh  
End_Procedure
```

XBase++

```
#include "AppEvent.ch"  
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

LOCAL oForm

LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

LOCAL oltems

LOCAL oRibbon

oForm := XbpDialog():new(AppDesktop())

oForm:drawingArea:clipChildren := .T.

oForm:create(,, {100,100}, {640,480},,, .F.)

oForm:close := {|| PostAppEvent(xbeP_Quit)}

oRibbon := XbpActiveXControl():new(oForm:drawingArea)

oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-CFBE431702E2}*/

oRibbon:create(,, {10,60},{610,370})

oltems := oRibbon:Items()

oltems:Add("Item"):Italic := .T.

oltems:Add("<i>Item</i>")

oltems:Add(""):ToString := "Item[itl]"

oRibbon:Refresh()

oForm:Show()

DO WHILE nEvent != xbeP_Quit

nEvent := AppEvent(@mp1, @mp2, @oXbp)

oXbp:handleEvent(nEvent, mp1, mp2)

ENDDO

RETURN

property Item.ItemHeight as Long

Specifies the fixed height to display the item.

Type	Description
Long	A Long expression that specifies the height of the item

The ItemHeight property specifies the height to display the item. By default, the ItemHeight property is -1, which indicates that the control sets the item height based on on its content. If the ItemHeight property is positive, it indicates the height to display the item. The [CaptionWidth](#) property specifies the fixed width to display the item's caption.

How can I specify the height of the item?

VBA (MS Access, Excell...)

```
With Ribbon1
    With .Items
        .BackColor = RGB(250,250,250)
        .Add("Item").ItemHeight = 64
        .Add("").ToString = "Item[height=64]"
    End With
    .Refresh
End With
```

VB6

```
With Ribbon1
    With .Items
        .BackColor = RGB(250,250,250)
        .Add("Item").ItemHeight = 64
        .Add("").ToString = "Item[height=64]"
    End With
    .Refresh
End With
```

VB.NET

```
With Exribbon1
    With .Items
        .BackColor = Color.FromArgb(250,250,250)
```

```

        .Add("Item").ItemHeight = 64
        .Add("").ToString = "Item[height=64]"
    End With
    .Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    With .Items
        .BackColor = RGB(250,250,250)
        .Add("Item").ItemHeight = 64
        .Add("").ToString = "Item[height=64]"
    End With
    .Refresh()
End With

```

C++

```

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
var_Items->PutBackColor(RGB(250,250,250));
var_Items->Add(L"Item",vtMissing,vtMissing)->PutItemHeight(64);
var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"Item[height=64]");
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
var_Items->BackColor = RGB(250,250,250);
var_Items->Add(L"Item",TNoParam(),TNoParam())->ItemHeight = 64;
var_Items->Add(L"",TNoParam(),TNoParam())->ToString = L"Item[height=64]";
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
var_Items.BackColor = Color.FromArgb(250,250,250);
var_Items.Add("Item",null,null).ItemHeight = 64;
var_Items.Add("",null,null).ToString = "Item[height=64]";
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var_Items.BackColor = 16448250;
    var_Items.Add("Item",null,null).ItemHeight = 64;
    var_Items.Add("",null,null).ToString = "Item[height=64]";
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>

```

```

<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Ribbon1
    With .Items
      .BackColor = RGB(250,250,250)
      .Add("Item").ItemHeight = 64
      .Add("").ToString = "Item[height=64]"
    End With
  .Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.BackColor =
(uint)ColorTranslator.ToWin32(Color.FromArgb(250,250,250));
var_Items.Add("Item",null,null).ItemHeight = 64;
var_Items.Add("",null,null).ToString = "Item[height=64]";
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
  COM com_Item,com_Items;
  anytype var_Item,var_Items;
  ;

  super();
}

```



```

var_Items = exribbon1.Items(); com_Items = var_Items;
com_Items.BackColor(WinApi::RGB2int(250,250,250));
var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
com_Item.ItemHeight(64);
var_Item = COM::createFromObject(com_Items.Add("")); com_Item = var_Item;
com_Item.ToString("Item[height=64]");
exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
  with Items do
  begin
    BackColor := $fafafa;
    Add('Item',Nil,Nil).ItemHeight := 64;
    Add('',Nil,Nil).ToString := 'Item[height=64]';
  end;
  Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  with Items do
  begin
    BackColor := $fafafa;
    Add('Item',Null,Null).ItemHeight := 64;
    Add('',Null,Null).ToString := 'Item[height=64]';
  end;
  Refresh();
end

```

```

with thisform.Ribbon1
  with .Items
    .BackColor = RGB(250,250,250)
    .Add("Item").ItemHeight = 64
    .Add("").ToString = "Item[height=64]"
  endwith
  .Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Items

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
var_Items.BackColor = 0xfafafa
// var_Items.Add("Item").ItemHeight = 64
var_Item = var_Items.Add("Item")
with (oRibbon)
  TemplateDef = [Dim var_Item]
  TemplateDef = var_Item
  Template = [var_Item.ItemHeight = 64]
endwith
// var_Items.Add("").ToString = "Item[height=64]"
var_Item1 = var_Items.Add("")
with (oRibbon)
  TemplateDef = [Dim var_Item1]
  TemplateDef = var_Item1
  Template = [var_Item1.ToString = "Item[height=64]"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P

```

```
Dim var_Items as P
```

```
oRibbon = topparent:CONTROL_ACTIVEX1.activex
```

```
var_Items = oRibbon.Items
```

```
var_Items.BackColor = 16448250
```

```
' var_Items.Add("Item").ItemHeight = 64
```

```
var_Item = var_Items.Add("Item")
```

```
oRibbon.TemplateDef = "Dim var_Item"
```

```
oRibbon.TemplateDef = var_Item
```

```
oRibbon.Template = "var_Item.ItemHeight = 64"
```

```
' var_Items.Add("").ToString = "Item[height=64]"
```

```
var_Item1 = var_Items.Add("")
```

```
oRibbon.TemplateDef = "Dim var_Item1"
```

```
oRibbon.TemplateDef = var_Item1
```

```
oRibbon.Template = "var_Item1.ToString = \"Item[height=64]\""
```

```
oRibbon.Refresh()
```

Visual Objects

```
local var_Items as IItems
```

```
var_Items := oDCOCX_Exontrol1:Items
```

```
var_Items.BackColor := RGB(250,250,250)
```

```
var_Items.Add("Item",nil,nil):ItemHeight := 64
```

```
var_Items.Add("",nil,nil):ToString := "Item[height=64]"
```

```
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Items
```

```
oRibbon = ole_1.Object
```

```
var_Items = oRibbon.Items
```

```
var_Items.BackColor = RGB(250,250,250)
```

```
var_Items.Add("Item").ItemHeight = 64
var_Items.Add("").ToString = "Item[height=64]"
oRibbon.Refresh()
```

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Variant voltems

Get ComItems to voltems

Handle holtems

Get Create (RefClass(cComItems)) to holtems

Set pvComObject of holtems to voltems

Set ComBackColor of holtems to (RGB(250,250,250))

Variant voltem

Get ComAdd of holtems "Item" Nothing Nothing to voltem

Handle holtem

Get Create (RefClass(cComItem)) to holtem

Set pvComObject of holtem to voltem

Set **ComItemHeight** of holtem to 64

Send Destroy to holtem

Variant voltem1

Get ComAdd of holtems "" Nothing Nothing to voltem1

Handle holtem1

Get Create (RefClass(cComItem)) to holtem1

Set pvComObject of holtem1 to voltem1

Set ComToString of holtem1 to "Item[height=64]"

Send Destroy to holtem1

Send Destroy to holtems

Send ComRefresh

End_Procedure

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"
```

PROCEDURE Main

LOCAL oForm

LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

LOCAL oltems

LOCAL oRibbon

oForm := XbpDialog():new(AppDesktop())

oForm:drawingArea:clipChildren := .T.

oForm:create(,, {100,100}, {640,480},,, .F.)

oForm:close := {|| PostAppEvent(xbeP_Quit)}

oRibbon := XbpActiveXControl():new(oForm:drawingArea)

oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-CFBE431702E2}*/

oRibbon:create(,, {10,60},{610,370})

oltems := oRibbon:Items()

oltems:SetProperty("BackColor",AutomationTranslateColor(
GraMakeRGBColor ({ 250,250,250 }) , .F.))

oltems:Add("Item"):ItemHeight := 64

oltems:Add(""):ToString := "Item[height=64]"

oRibbon:Refresh()

oForm:Show()

DO WHILE nEvent != xbeP_Quit

nEvent := AppEvent(@mp1, @mp2, @oXbp)

oXbp:handleEvent(nEvent, mp1, mp2)

ENDDO

RETURN

property Item.Items as Items

Retrieves an Items collection that indicates the item's sub menu. Retrieves Nothing, if the item contains no sub menu.

Type	Description
Items	An Items collection that holds the Item objects to be displayed in the sub-menu.

The Items and [SubMenu](#) properties are equivalents. Use the Items property to access the Item objects in a sub-menu item. The [Parent](#) property of the Item object returns an empty object, if the item contains no parent. The Parent item property can be used to access the parent of the item, when it is contained by a sub-menu.

property Item.ItemType as ItemTypeEnum

Returns the type of the item.

Type	Description
ItemTypeEnum	An ItemTypeEnum expression that specifies the type of the item.

The ItemType property is a read-only property that gets the type of the item. Use the [Debug](#) property to display debugging information in the item's Caption. Use the [Get](#) method to get a collection of Item objects that meet your criteria.

property Item.Padding as String

Specifies the padding (space between the menu border and the item content) to display the item.

Type	Description
String	A string expression that indicates a list of 4 positive numbers separated by comma characters, which indicates the distance in pixels from margin to client, in the following format: left, top, right, bottom.

By default, the Padding property is empty "0,0,0,0". The Padding property specifies the padding for a particular item. The Padding property specifies the padding (space between the menu border and the item content) to display the item. The [Caption](#) property indicates the item's caption to be shown on the item. The [BackColor](#) property specifies a different background color or a visual appearance for the item. The [Padding](#) property of the Items collection specifies the padding for all items in the collection.

Does your control implement padding for item?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    .BackColor = RGB(250,250,250)
    .Add("Item").Padding = "4,4,4,4"
    .Add("").ToString = "Item[pad=4,4,4,4]"
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    .BackColor = RGB(250,250,250)
    .Add("Item").Padding = "4,4,4,4"
    .Add("").ToString = "Item[pad=4,4,4,4]"
  End With
  .Refresh
End With
```


VB.NET

```
With Exribbon1
    With .Items
        .BackColor = Color.FromArgb(250,250,250)
        .Add("Item").Padding = "4,4,4,4"
        .Add("").ToString = "Item[pad=4,4,4,4]"
    End With
    .Refresh()
End With
```

VB.NET for /COM

```
With AxRibbon1
    With .Items
        .BackColor = RGB(250,250,250)
        .Add("Item").Padding = "4,4,4,4"
        .Add("").ToString = "Item[pad=4,4,4,4]"
    End With
    .Refresh()
End With
```

C++

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
> GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
var_Items->PutBackColor(RGB(250,250,250));
var_Items->Add(L"Item",vtMissing,vtMissing)->PutPadding(L"4,4,4,4");
var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"Item[pad=4,4,4,4]");
```

```
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;  
var_Items->BackColor = RGB(250,250,250);  
var_Items->Add(L"Item",TNoParam(),TNoParam())->Padding = L"4,4,4,4";  
var_Items->Add(L"",TNoParam(),TNoParam())->ToString = L"Item[pad=4,4,4,4]";  
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
var_Items.BackColor = Color.FromArgb(250,250,250);  
var_Items.Add("Item",null,null).Padding = "4,4,4,4";  
var_Items.Add("",null,null).ToString = "Item[pad=4,4,4,4]";  
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    var var_Items = Ribbon1.Items;  
    var_Items.BackColor = 16448250;  
    var_Items.Add("Item",null,null).Padding = "4,4,4,4";  
    var_Items.Add("",null,null).ToString = "Item[pad=4,4,4,4]";  
    Ribbon1.Refresh();  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload='Init()>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"></OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
  With Ribbon1  
    With .Items  
      .BackColor = RGB(250,250,250)  
      .Add("Item").Padding = "4,4,4,4"  
      .Add("").ToString = "Item[pad=4,4,4,4]"  
    End With  
    .Refresh  
  End With  
End Function  
</SCRIPT>  
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;  
var_Items.BackColor =  
(uint)ColorTranslator.ToWin32(Color.FromArgb(250,250,250));  
var_Items.Add("Item",null,null).Padding = "4,4,4,4";  
var_Items.Add("",null,null).ToString = "Item[pad=4,4,4,4]";  
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()  
{  
  COM com_Item,com_Items;
```

```

anytype var_Item,var_Items;
;

super();

var_Items = exribbon1.Items(); com_Items = var_Items;
    com_Items.BackColor(WinApi::RGB2int(250,250,250));
    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
    com_Item.Padding("4,4,4,4");
    var_Item = COM::createFromObject(com_Items.Add("")); com_Item = var_Item;
    com_Item.ToString("Item[pad=4,4,4,4]");
exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        BackColor := $fafafa;
        Add('Item',Nil,Nil).Padding := '4,4,4,4';
        Add('',Nil,Nil).ToString := 'Item[pad=4,4,4,4]';
    end;
    Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin
        BackColor := $fafafa;
        Add('Item',Null,Null).Padding := '4,4,4,4';
        Add('',Null,Null).ToString := 'Item[pad=4,4,4,4]';
    end;
end;

```

```
Refresh();  
end
```

VFP

```
with thisform.Ribbon1  
  with .Items  
    .BackColor = RGB(250,250,250)  
    .Add("Item").Padding = "4,4,4,4"  
    .Add("").ToString = "Item[pad=4,4,4,4]"  
  endwhile  
  .Refresh  
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Item1,var_Items  
  
oRibbon = form.Activex1.nativeObject  
var_Items = oRibbon.Items  
var_Items.BackColor = 0xfafafa  
// var_Items.Add("Item").Padding = "4,4,4,4"  
var_Item = var_Items.Add("Item")  
with (oRibbon)  
  TemplateDef = [Dim var_Item]  
  TemplateDef = var_Item  
  Template = [var_Item.Padding = "4,4,4,4"]  
endwith  
// var_Items.Add("").ToString = "Item[pad=4,4,4,4]"  
var_Item1 = var_Items.Add("")  
with (oRibbon)  
  TemplateDef = [Dim var_Item1]  
  TemplateDef = var_Item1  
  Template = [var_Item1.ToString = "Item[pad=4,4,4,4]"]  
endwith  
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Items.BackColor = 16448250
' var_Items.Add("Item").Padding = "4,4,4,4"
var_Item = var_Items.Add("Item")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.Padding = \"4,4,4,4\""

' var_Items.Add("").ToString = "Item[pad=4,4,4,4]"
var_Item1 = var_Items.Add("")
oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1
oRibbon.Template = "var_Item1.ToString = \"Item[pad=4,4,4,4]\""

oRibbon.Refresh()
```

Visual Objects

```
local var_Items as IItems

var_Items := oDCOCX_Exontrol1.Items
var_Items.BackColor := RGB(250,250,250)
var_Items.Add("Item",nil,nil):Padding := "4,4,4,4"
var_Items.Add("",nil,nil):ToString := "Item[pad=4,4,4,4]"
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

OleObject oRibbon,var_Items

oRibbon = ole_1.Object

var_Items = oRibbon.Items

var_Items.BackColor = RGB(250,250,250)

var_Items.Add("Item").**Padding** = "4,4,4,4"

var_Items.Add("").ToString = "Item[pad=4,4,4,4]"

oRibbon.Refresh()

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Variant voltems

Get Comltems to voltems

Handle holtems

Get Create (RefClass(cComltems)) to holtems

Set pvComObject of holtems to voltems

Set ComBackColor of holtems to (RGB(250,250,250))

Variant voltem

Get ComAdd of holtems "Item" Nothing Nothing to voltem

Handle holtem

Get Create (RefClass(cComltem)) to holtem

Set pvComObject of holtem to voltem

Set **ComPadding** of holtem to "4,4,4,4"

Send Destroy to holtem

Variant voltem1

Get ComAdd of holtems "" Nothing Nothing to voltem1

Handle holtem1

Get Create (RefClass(cComltem)) to holtem1

Set pvComObject of holtem1 to voltem1

Set ComToString of holtem1 to "Item[pad=4,4,4,4]"

Send Destroy to holtem1

Send Destroy to holtems

Send ComRefresh

End_Procedure

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oltems := oRibbon:Items()
    oltems:SetProperty("BackColor",AutomationTranslateColor(
GraMakeRGBColor ( { 250,250,250 } ) , .F. ))
    oltems:Add("Item"):Padding := "4,4,4,4"
    oltems:Add(""):ToString := "Item[pad=4,4,4,4]"
    oRibbon:Refresh()

    oForm:Show()
    DO WHILE nEvent != xbeP_Quit
        nEvent := AppEvent( @mp1, @mp2, @oXbp )
        oXbp:handleEvent( nEvent, mp1, mp2 )
    ENDDO
    RETURN
```


property Item.Parent as Item

Gets the item's parent, if the current item belongs to a submenu/popup.

Type	Description
Item	An Item object that specifies the parent item of the current item.

The Parent property of the Item object returns an empty object, if the item contains no parent. The Parent item property can be used to access the parent of the item, when it is contained by a sub-menu. Use the [Items](#) property to access the Item objects in a sub-menu item.

property Item.Position as Long

Specifies the position of the item, within its collection.

Type	Description
Long	A long expression that specifies the position of the item (0-based)

The Position property specifies the position of the item, within its collection. You can use the Position property to change the position of specified item. The [Count](#) property returns the number of objects in a collection. The [Visible](#) property specifies whether the item is visible or hidden.

property Item.Radio as Boolean

Retrieves or sets a value that indicates whether the item is of radio type.

Type	Description
Boolean	A Boolean expression that specifies whether the item displays a radio button.

The Radio property specifies whether the item displays a radio-button. The [RadioGroup](#) property specifies a group of radio-buttons. A radio group allows a single radio-item to be checked. The [Checked](#) property specifies whether the item is checked or un-checked. The [GetRadio](#) method gets a safe array with the radio-items being checked within a radio group. Use the [Background\(exRadioButtonState0\)/Background\(exRadioButtonState1\)](#) property to specify the visual appearance of the radio-buttons in the control. Use the [UseVisualTheme](#) property to specify whether the visual appearance for the radio-buttons to be as indicated by the current XP theme. The [AllowToggleRadio](#) property on True, allows a radio button to set on zero (unchecked), if the user clicks twice the radio button. Usually, clicking a radio-button makes the previously checked radio-button in the same group, to be un-checked, and the newly clicked item to be checked. Now, if the AllowToggleRadio property is True, clicking again the radio-button, allows the radio-button to be un-checked, so allows a radio group to have no radio button checked. Use the [ShowCheckedAsSelected](#) property on True, to show the checked items as selected.

How can I add radio buttons to items?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("",2)
      .GroupPopup = 1
    With .Items
      With .Add("Radio 1",0,1000)
        .Radio = True
        .RadioGroup = 100
      End With
    With .Add("Radio 2",0,1001)
      .Radio = True
      .RadioGroup = 100
    End With
  With .Add("Radio 2",0,1003)
```

```
.Radio = True
.RadioGroup = 100
End With
.Item(1000).Checked = True
End With
End With
End With
.Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    With .Add("",2)
      .GroupPopup = exGroupPopup
    With .Items
      With .Add("Radio 1",0,1000)
        .Radio = True
        .RadioGroup = 100
      End With
      With .Add("Radio 2",0,1001)
        .Radio = True
        .RadioGroup = 100
      End With
      With .Add("Radio 2",0,1003)
        .Radio = True
        .RadioGroup = 100
      End With
      .Item(1000).Checked = True
    End With
  End With
End With
.Refresh
End With
```

VB.NET

With ExRibbon1

With .Items

With **.Add**("",2)

.GroupPopup = exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup

With .Items

With **.Add**("Radio 1",0,1000)

.Radio = True

.RadioGroup = 100

End With

With **.Add**("Radio 2",0,1001)

.Radio = True

.RadioGroup = 100

End With

With **.Add**("Radio 2",0,1003)

.Radio = True

.RadioGroup = 100

End With

.Item(1000).Checked = True

End With

End With

End With

.Refresh()

End With

VB.NET for /COM

With AxRibbon1

With .Items

With **.Add**("",2)

.GroupPopup = EXRIBBONLib.GroupPopupEnum.exGroupPopup

With .Items

With **.Add**("Radio 1",0,1000)

.Radio = True

.RadioGroup = 100

End With

With **.Add**("Radio 2",0,1001)

.Radio = True

```

        .RadioGroup = 100
    End With
    With .Add("Radio 2",0,1003)
        .Radio = True
        .RadioGroup = 100
    End With
    .Item(1000).Checked = True
End With
End With
End With
.Refresh()
End With

```

C++

```

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

#import <ExRibbon.dll>
using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"",long(2),vtMissing);
var_Item->PutGroupPopup(EXRIBBONLib::exGroupPopup);
EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();
EXRIBBONLib::IItemPtr var_Item1 = var_Items1->Add(L"Radio
1",long(0),long(1000));
var_Item1->PutRadio(VARIANT_TRUE);
var_Item1->PutRadioGroup(100);
EXRIBBONLib::IItemPtr var_Item2 = var_Items1->Add(L"Radio
2",long(0),long(1001));
var_Item2->PutRadio(VARIANT_TRUE);
var_Item2->PutRadioGroup(100);

```

```

    EXRIBBONLib::ItemPtr var_Item3 = var_Items1->Add(L"Radio
2",long(0),long(1003));
    var_Item3->PutRadio(VARIANT_TRUE);
    var_Item3->PutRadioGroup(100);
    var_Items1->GetItem(long(1000))->PutChecked(VARIANT_TRUE);
    spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::ItemsPtr var_Items = Ribbon1->Items;
Exribbonlib_tlb::ItemPtr var_Item = var_Items->Add(L"",TVariant(2),TNoParam());
var_Item->GroupPopup = Exribbonlib_tlb::GroupPopupEnum::exGroupPopup;
Exribbonlib_tlb::ItemsPtr var_Items1 = var_Item->Items;
    Exribbonlib_tlb::ItemPtr var_Item1 = var_Items1->Add(L"Radio
1",TVariant(0),TVariant(1000));
    var_Item1->Radio = true;
    var_Item1->RadioGroup = 100;
    Exribbonlib_tlb::ItemPtr var_Item2 = var_Items1->Add(L"Radio
2",TVariant(0),TVariant(1001));
    var_Item2->Radio = true;
    var_Item2->RadioGroup = 100;
    Exribbonlib_tlb::ItemPtr var_Item3 = var_Items1->Add(L"Radio
2",TVariant(0),TVariant(1003));
    var_Item3->Radio = true;
    var_Item3->RadioGroup = 100;
    var_Items1->get_Item(TVariant(1000))->Checked = true;
    Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
    exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
    var_Item.GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup;
    exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
    exontrol.EXRIBBONLib.Item var_Item1 = var_Items1.Add("Radio 1",0,1000);

```

```

var_Item1.Radio = true;
var_Item1.RadioGroup = 100;
exontrol.EXRIBBONLib.Item var_Item2 = var_Items1.Add("Radio 2",0,1001);
var_Item2.Radio = true;
var_Item2.RadioGroup = 100;
exontrol.EXRIBBONLib.Item var_Item3 = var_Items1.Add("Radio 2",0,1003);
var_Item3.Radio = true;
var_Item3.RadioGroup = 100;
var_Items1[1000].Checked = true;
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var var_Item = var_Items.Add("",2,null);
    var_Item.GroupPopup = 1;
    var var_Items1 = var_Item.Items;
    var var_Item1 = var_Items1.Add("Radio 1",0,1000);
    var_Item1.Radio = true;
    var_Item1.RadioGroup = 100;
    var var_Item2 = var_Items1.Add("Radio 2",0,1001);
    var_Item2.Radio = true;
    var_Item2.RadioGroup = 100;
    var var_Item3 = var_Items1.Add("Radio 2",0,1003);
    var_Item3.Radio = true;
    var_Item3.RadioGroup = 100;
    var_Items1.Item(1000).Checked = true;
    Ribbon1.Refresh();
}

```



```
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
  With Ribbon1  
    With .Items  
      With .Add("",2)  
        .GroupPopup = 1  
        With .Items  
          With .Add("Radio 1",0,1000)  
            .Radio = True  
            .RadioGroup = 100  
          End With  
          With .Add("Radio 2",0,1001)  
            .Radio = True  
            .RadioGroup = 100  
          End With  
          With .Add("Radio 2",0,1003)  
            .Radio = True  
            .RadioGroup = 100  
          End With  
          .Item(1000).Checked = True  
        End With  
      End With  
    End With  
  End With  
  .Refresh  
End With  
End Function  
</SCRIPT>
```

</BODY>

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;
EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
var_Item.GroupPopup = EXRIBBONLib.GroupPopupEnum.exGroupPopup;
EXRIBBONLib.Items var_Items1 = var_Item.Items;
EXRIBBONLib.Item var_Item1 = var_Items1.Add("Radio 1",0,1000);
var_Item1.Radio = true;
var_Item1.RadioGroup = 100;
EXRIBBONLib.Item var_Item2 = var_Items1.Add("Radio 2",0,1001);
var_Item2.Radio = true;
var_Item2.RadioGroup = 100;
EXRIBBONLib.Item var_Item3 = var_Items1.Add("Radio 2",0,1003);
var_Item3.Radio = true;
var_Item3.RadioGroup = 100;
var_Items1[1000].Checked = true;
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM
    com_Item,com_Item1,com_Item2,com_Item3,com_Item4,com_Items,com_Items1;
    anytype var_Item,var_Item1,var_Item2,var_Item3,var_Item4,var_Items,var_Items1;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("",COMVariant::createFromInt(2)); com_Item =
    var_Item;
    com_Item.GroupPopup(1/*exGroupPopup*/);
    var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
```

```

        var_Item1 = com_Items1.Add("Radio
1",COMVariant::createFromInt(0),COMVariant::createFromInt(1000)); com_Item1 =
var_Item1;
        com_Item1.Radio(true);
        com_Item1.RadioGroup(100);
        var_Item2 = com_Items1.Add("Radio
2",COMVariant::createFromInt(0),COMVariant::createFromInt(1001)); com_Item2 =
var_Item2;
        com_Item2.Radio(true);
        com_Item2.RadioGroup(100);
        var_Item3 = com_Items1.Add("Radio
2",COMVariant::createFromInt(0),COMVariant::createFromInt(1003)); com_Item3 =
var_Item3;
        com_Item3.Radio(true);
        com_Item3.RadioGroup(100);
        var_Item4 =
COM::createFromObject(com_Items1.Item(COMVariant::createFromInt(1000)));
com_Item4 = var_Item4;
        com_Item4.Checked(true);
        exribbon1.Refresh();
    }

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        with Add('',TObject(2),Nil) do
        begin
            GroupPopup := EXRIBBONLib.GroupPopupEnum.exGroupPopup;
            with Items do
            begin
                with Add('Radio 1',TObject(0),TObject(1000)) do
                begin
                    Radio := True;
                    RadioGroup := 100;

```

```

end;
with Add('Radio 2',TObject(0),TObject(1001)) do
begin
    Radio := True;
    RadioGroup := 100;
end;
with Add('Radio 2',TObject(0),TObject(1003)) do
begin
    Radio := True;
    RadioGroup := 100;
end;
Item[TObject(1000)].Checked := True;
end;
end;
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin
        with Add('',OleVariant(2),Null) do
        begin
            GroupPopup := EXRIBBONLib_TLB.exGroupPopup;
            with Items do
            begin
                with Add('Radio 1',OleVariant(0),OleVariant(1000)) do
                begin
                    Radio := True;
                    RadioGroup := 100;
                end;
                with Add('Radio 2',OleVariant(0),OleVariant(1001)) do
                begin
                    Radio := True;

```

```

        RadioGroup := 100;
    end;
    with Add('Radio 2',OleVariant(0),OleVariant(1003)) do
    begin
        Radio := True;
        RadioGroup := 100;
    end;
    Item[OleVariant(1000)].Checked := True;
end;
end;
end;
Refresh();
end

```

VFP

```

with thisform.Ribbon1
with .Items
with .Add("",2)
    .GroupPopup = 1
    with .Items
        with .Add("Radio 1",0,1000)
            .Radio = .T.
            .RadioGroup = 100
        endwith
        with .Add("Radio 2",0,1001)
            .Radio = .T.
            .RadioGroup = 100
        endwith
        with .Add("Radio 2",0,1003)
            .Radio = .T.
            .RadioGroup = 100
        endwith
        .Item(1000).Checked = .T.
    endwith
endwith
endwith
endwith

```

.Refresh
endwith

dBASE Plus

```
local oRibbon,var_Item,var_Item1,var_Item2,var_Item3,var_Item4,var_Items,var_Items1

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 1
var_Items1 = var_Item.Items
var_Item1 = var_Items1.Add("Radio 1",0,1000)
var_Item1.Radio = true
var_Item1.RadioGroup = 100
var_Item2 = var_Items1.Add("Radio 2",0,1001)
var_Item2.Radio = true
var_Item2.RadioGroup = 100
var_Item3 = var_Items1.Add("Radio 2",0,1003)
var_Item3.Radio = true
var_Item3.RadioGroup = 100
// var_Items1.Item(1000).Checked = true
var_Item4 = var_Items1.Item(1000)
with (oRibbon)
    TemplateDef = [Dim var_Item4]
    TemplateDef = var_Item4
    Template = [var_Item4.Checked = true]
endwith
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Item2 as P
Dim var_Item3 as P
```

```

Dim var_Item4 as P
Dim var_Items as P
Dim var_Items1 as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 1
var_Items1 = var_Item.Items
var_Item1 = var_Items1.Add("Radio 1",0,1000)
var_Item1.Radio = .t.
var_Item1.RadioGroup = 100
var_Item2 = var_Items1.Add("Radio 2",0,1001)
var_Item2.Radio = .t.
var_Item2.RadioGroup = 100
var_Item3 = var_Items1.Add("Radio 2",0,1003)
var_Item3.Radio = .t.
var_Item3.RadioGroup = 100
' var_Items1.Item(1000).Checked = .t.
var_Item4 = var_Items1.Item(1000)
oRibbon.TemplateDef = "Dim var_Item4"
oRibbon.TemplateDef = var_Item4
oRibbon.Template = "var_Item4.Checked = True"

oRibbon.Refresh()

```

Visual Objects

```

local var_Item,var_Item1,var_Item2,var_Item3 as IItem
local var_Items,var_Items1 as IItems

var_Items := oDCOCX_Exontrol1:Items
var_Item := var_Items.Add("",2,nil)
var_Item:GroupPopup := exGroupPopup
var_Items1 := var_Item:Items
var_Item1 := var_Items1.Add("Radio 1",0,1000)

```

```

    var_Item1:Radio := true
    var_Item1:RadioGroup := 100
    var_Item2 := var_Items1:Add("Radio 2",0,1001)
    var_Item2:Radio := true
    var_Item2:RadioGroup := 100
    var_Item3 := var_Items1:Add("Radio 2",0,1003)
    var_Item3:Radio := true
    var_Item3:RadioGroup := 100
    var_Items1:[Item,1000]:Checked := true
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```

OleObject oRibbon,var_Item,var_Item1,var_Item2,var_Item3,var_Items,var_Items1

oRibbon = ole_1.Object
var_Items = oRibbon.Items
    var_Item = var_Items.Add("",2)
    var_Item.GroupPopup = 1
    var_Items1 = var_Item.Items
    var_Item1 = var_Items1.Add("Radio 1",0,1000)
    var_Item1.Radio = true
    var_Item1.RadioGroup = 100
    var_Item2 = var_Items1.Add("Radio 2",0,1001)
    var_Item2.Radio = true
    var_Item2.RadioGroup = 100
    var_Item3 = var_Items1.Add("Radio 2",0,1003)
    var_Item3.Radio = true
    var_Item3.RadioGroup = 100
    var_Items1.Item(1000).Checked = true
oRibbon.Refresh()

```

Visual DataFlex

```

Procedure OnCreate
    Forward Send OnCreate

```


Variant voltems

Get Comltems to voltems

Handle holtems

Get Create (RefClass(cComltems)) to holtems

Set pvComObject of holtems to voltems

Variant voltem

Get **ComAdd** of holtems "" 2 Nothing to voltem

Handle holtem

Get Create (RefClass(cComltem)) to holtem

Set pvComObject of holtem to voltem

Set ComGroupPopup of holtem to OLEexGroupPopup

Variant voltems1

Get Comltems of holtem to voltems1

Handle holtems1

Get Create (RefClass(cComltems)) to holtems1

Set pvComObject of holtems1 to voltems1

Variant voltem1

Get **ComAdd** of holtems1 "Radio 1" 0 1000 to voltem1

Handle holtem1

Get Create (RefClass(cComltem)) to holtem1

Set pvComObject of holtem1 to voltem1

Set **ComRadio** of holtem1 to True

Set **ComRadioGroup** of holtem1 to 100

Send Destroy to holtem1

Variant voltem2

Get **ComAdd** of holtems1 "Radio 2" 0 1001 to voltem2

Handle holtem2

Get Create (RefClass(cComltem)) to holtem2

Set pvComObject of holtem2 to voltem2

Set **ComRadio** of holtem2 to True

Set **ComRadioGroup** of holtem2 to 100

Send Destroy to holtem2

Variant voltem3

Get **ComAdd** of holtems1 "Radio 2" 0 1003 to voltem3

Handle holtem3

Get Create (RefClass(cComltem)) to holtem3

Set pvComObject of holtem3 to voltem3

```

        Set ComRadio of holtem3 to True
        Set ComRadioGroup of holtem3 to 100
    Send Destroy to holtem3
    Variant voltem4
    Get ComItem of holtems1 1000 to voltem4
    Handle holtem4
    Get Create (RefClass(cComItem)) to holtem4
    Set pvComObject of holtem4 to voltem4
        Set ComChecked of holtem4 to True
    Send Destroy to holtem4
    Send Destroy to holtems1
    Send Destroy to holtem
    Send Destroy to holtems
    Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem,oltem1,oltem2,oltem3
    LOCAL oltems,oltems1
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

```

```
oItems := oRibbon:Items()
oItem := oItems:Add("",2)
oItem:GroupPopup := 1/*exGroupPopup*/
oItems1 := oItem:Items()
  oItem1 := oItems1:Add("Radio 1",0,1000)
  oItem1:Radio := .T.
  oItem1:RadioGroup := 100
  oItem2 := oItems1:Add("Radio 2",0,1001)
  oItem2:Radio := .T.
  oItem2:RadioGroup := 100
  oItem3 := oItems1:Add("Radio 2",0,1003)
  oItem3:Radio := .T.
  oItem3:RadioGroup := 100
  oItems1:Item(1000):Checked := .T.
oRibbon:Refresh()
```

```
oForm:Show()
DO WHILE nEvent != xbeP_Quit
  nEvent := AppEvent( @mp1, @mp2, @oXbp )
  oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN
```

property Item.RadioGroup as Long

Indicates the group of radio items that the current item belongs.

Type	Description
Long	A Long expression that specifies the identifier of the radio group. A radio group allows a single radio-item to be checked inside. If the RadioGroup property is not specified, all radio-buttons in the controls are in the same group 0.

The RadioGroup property specifies a group of radio-buttons. A radio group allows a single radio-item to be checked inside. The [Radio](#) property specifies whether the item displays a radio-button. The [Checked](#) property specifies whether the item is checked or un-checked. The [GetRadio](#) method gets a safe array with the radio-items being checked within a radio group. Use the [Background\(exRadioButtonState0\)/Background\(exRadioButtonState1\)](#) property to specify the visual appearance of the radio-buttons in the control. Use the [UseVisualTheme](#) property to specify whether the visual appearance for the radio-buttons to be as indicated by the current XP theme. The [AllowToggleRadio](#) property on True, allows a radio button to set on zero (unchecked), if the user clicks twice the radio button. Usually, clicking a radio-button makes the previously checked radio-button in the same group, to be un-checked, and the newly clicked item to be checked. Now, if the AllowToggleRadio property is True, clicking again the radio-button, allows the radio-button to be un-checked, so allows a radio group to have no radio button checked.

I am using radio-buttons, the question is it possible to uncheck the radio-buttons, so no button is pressed in the group?

VBA (MS Access, Excell...)

```
With Ribbon1
    .AllowToggleRadio = True
    With .Items
        With .Add("Radio 1",0,1000)
            .Radio = True
            .RadioGroup = 100
        End With
        With .Add("Radio 2",0,1001)
            .Radio = True
            .RadioGroup = 100
        End With
        With .Add("Radio 2",0,1003)
```

```
.Radio = True
.RadioGroup = 100
End With
End With
.Refresh
End With
```

VB6

```
With Ribbon1
.AllowToggleRadio = True
With .Items
With .Add("Radio 1",0,1000)
.Radio = True
.RadioGroup = 100
End With
With .Add("Radio 2",0,1001)
.Radio = True
.RadioGroup = 100
End With
With .Add("Radio 2",0,1003)
.Radio = True
.RadioGroup = 100
End With
End With
.Refresh
End With
```

VB.NET

```
With Exribbon1
.AllowToggleRadio = True
With .Items
With .Add("Radio 1",0,1000)
.Radio = True
.RadioGroup = 100
End With
With .Add("Radio 2",0,1001)
```

```

        .Radio = True
        .RadioGroup = 100
    End With
    With .Add("Radio 2",0,1003)
        .Radio = True
        .RadioGroup = 100
    End With
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    .AllowToggleRadio = True
    With .Items
        With .Add("Radio 1",0,1000)
            .Radio = True
            .RadioGroup = 100
        End With
        With .Add("Radio 2",0,1001)
            .Radio = True
            .RadioGroup = 100
        End With
        With .Add("Radio 2",0,1003)
            .Radio = True
            .RadioGroup = 100
        End With
    End With
    .Refresh()
End With

```

C++

```

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

```

```

#import <ExRibbon.dll>
using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
spRibbon1->PutAllowToggleRadio(VARIANT_TRUE);
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Radio 1",long(0),long(1000));
    var_Item->PutRadio(VARIANT_TRUE);
    var_Item->PutRadioGroup(100);
    EXRIBBONLib::IItemPtr var_Item1 = var_Items->Add(L"Radio
2",long(0),long(1001));
    var_Item1->PutRadio(VARIANT_TRUE);
    var_Item1->PutRadioGroup(100);
    EXRIBBONLib::IItemPtr var_Item2 = var_Items->Add(L"Radio
2",long(0),long(1003));
    var_Item2->PutRadio(VARIANT_TRUE);
    var_Item2->PutRadioGroup(100);
spRibbon1->Refresh();

```

C++ Builder

```

Ribbon1->AllowToggleRadio = true;
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    Exribbonlib_tlb::IItemPtr var_Item = var_Items->Add(L"Radio
1",TVariant(0),TVariant(1000));
    var_Item->Radio = true;
    var_Item->RadioGroup = 100;
    Exribbonlib_tlb::IItemPtr var_Item1 = var_Items->Add(L"Radio
2",TVariant(0),TVariant(1001));
    var_Item1->Radio = true;
    var_Item1->RadioGroup = 100;
    Exribbonlib_tlb::IItemPtr var_Item2 = var_Items->Add(L"Radio
2",TVariant(0),TVariant(1003));
    var_Item2->Radio = true;

```

```
var_Item2->RadioGroup = 100;  
Ribbon1->Refresh();
```

C#

```
exribbon1.AllowToggleRadio = true;  
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Radio 1",0,1000);  
var_Item.Radio = true;  
var_Item.RadioGroup = 100;  
exontrol.EXRIBBONLib.Item var_Item1 = var_Items.Add("Radio 2",0,1001);  
var_Item1.Radio = true;  
var_Item1.RadioGroup = 100;  
exontrol.EXRIBBONLib.Item var_Item2 = var_Items.Add("Radio 2",0,1003);  
var_Item2.Radio = true;  
var_Item2.RadioGroup = 100;  
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
Ribbon1.AllowToggleRadio = true;  
var var_Items = Ribbon1.Items;  
var var_Item = var_Items.Add("Radio 1",0,1000);  
var_Item.Radio = true;  
var_Item.RadioGroup = 100;  
var var_Item1 = var_Items.Add("Radio 2",0,1001);  
var_Item1.Radio = true;  
var_Item1.RadioGroup = 100;  
var var_Item2 = var_Items.Add("Radio 2",0,1003);
```



```
var_Item2.Radio = true;
var_Item2.RadioGroup = 100;
Ribbon1.Refresh();
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Ribbon1
    .AllowToggleRadio = True
  With .Items
    With .Add("Radio 1",0,1000)
      .Radio = True
      .RadioGroup = 100
    End With
    With .Add("Radio 2",0,1001)
      .Radio = True
      .RadioGroup = 100
    End With
    With .Add("Radio 2",0,1003)
      .Radio = True
      .RadioGroup = 100
    End With
  End With
  .Refresh
End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
axRibbon1.AllowToggleRadio = true;
EXRIBBONLib.Items var_Items = axRibbon1.Items;
    EXRIBBONLib.Item var_Item = var_Items.Add("Radio 1",0,1000);
        var_Item.Radio = true;
        var_Item.RadioGroup = 100;
    EXRIBBONLib.Item var_Item1 = var_Items.Add("Radio 2",0,1001);
        var_Item1.Radio = true;
        var_Item1.RadioGroup = 100;
    EXRIBBONLib.Item var_Item2 = var_Items.Add("Radio 2",0,1003);
        var_Item2.Radio = true;
        var_Item2.RadioGroup = 100;
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Item,com_Item1,com_Item2,com_Items;
    anytype var_Item,var_Item1,var_Item2,var_Items;
    ;

    super();

    exribbon1.AllowToggleRadio(true);
    var_Items = exribbon1.Items(); com_Items = var_Items;
        var_Item = com_Items.Add("Radio
1",COMVariant::createFromInt(0),COMVariant::createFromInt(1000)); com_Item =
var_Item;
            com_Item.Radio(true);
            com_Item.RadioGroup(100);
        var_Item1 = com_Items.Add("Radio
2",COMVariant::createFromInt(0),COMVariant::createFromInt(1001)); com_Item1 =
var_Item1;
```

```

        com_Item1.Radio(true);
        com_Item1.RadioGroup(100);
        var_Item2 = com_Items.Add("Radio
2",COMVariant::createFromInt(0),COMVariant::createFromInt(1003)); com_Item2 =
var_Item2;
        com_Item2.Radio(true);
        com_Item2.RadioGroup(100);
        exribbon1.Refresh();
    }

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    AllowToggleRadio := True;
    with Items do
    begin
        with Add('Radio 1',TObject(0),TObject(1000)) do
        begin
            Radio := True;
            RadioGroup := 100;
        end;
        with Add('Radio 2',TObject(0),TObject(1001)) do
        begin
            Radio := True;
            RadioGroup := 100;
        end;
        with Add('Radio 2',TObject(0),TObject(1003)) do
        begin
            Radio := True;
            RadioGroup := 100;
        end;
    end;
    Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  AllowToggleRadio := True;
  with Items do
  begin
    with Add('Radio 1',OleVariant(0),OleVariant(1000)) do
    begin
      Radio := True;
      RadioGroup := 100;
    end;
    with Add('Radio 2',OleVariant(0),OleVariant(1001)) do
    begin
      Radio := True;
      RadioGroup := 100;
    end;
    with Add('Radio 2',OleVariant(0),OleVariant(1003)) do
    begin
      Radio := True;
      RadioGroup := 100;
    end;
  end;
  Refresh();
end

```

VFP

```

with thisform.Ribbon1
.AllowToggleRadio = .T.
with .Items
  with .Add("Radio 1",0,1000)
    .Radio = .T.
    .RadioGroup = 100
  endwith
  with .Add("Radio 2",0,1001)
    .Radio = .T.
    .RadioGroup = 100
  endwith

```

```
with .Add("Radio 2",0,1003)
    .Radio = .T.
    .RadioGroup = 100
endwith
endwith
.Refresh
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Item1,var_Item2,var_Items

oRibbon = form.Activex1.nativeObject
oRibbon.AllowToggleRadio = true
var_Items = oRibbon.Items
    var_Item = var_Items.Add("Radio 1",0,1000)
        var_Item.Radio = true
        var_Item.RadioGroup = 100
    var_Item1 = var_Items.Add("Radio 2",0,1001)
        var_Item1.Radio = true
        var_Item1.RadioGroup = 100
    var_Item2 = var_Items.Add("Radio 2",0,1003)
        var_Item2.Radio = true
        var_Item2.RadioGroup = 100
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Item2 as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
oRibbon.AllowToggleRadio = .t.
var_Items =oRibbon.Items
```

```

var_Item = var_Items.Add("Radio 1",0,1000)
    var_Item.Radio = .t.
    var_Item.RadioGroup = 100
var_Item1 = var_Items.Add("Radio 2",0,1001)
    var_Item1.Radio = .t.
    var_Item1.RadioGroup = 100
var_Item2 = var_Items.Add("Radio 2",0,1003)
    var_Item2.Radio = .t.
    var_Item2.RadioGroup = 100
oRibbon.Refresh()

```

Visual Objects

```

local var_Item,var_Item1,var_Item2 as Item
local var_Items as Items

oDCOCX_Exontrol1:AllowToggleRadio := true
var_Items := oDCOCX_Exontrol1:Items
    var_Item := var_Items.Add("Radio 1",0,1000)
        var_Item:Radio := true
        var_Item:RadioGroup := 100
    var_Item1 := var_Items.Add("Radio 2",0,1001)
        var_Item1:Radio := true
        var_Item1:RadioGroup := 100
    var_Item2 := var_Items.Add("Radio 2",0,1003)
        var_Item2:Radio := true
        var_Item2:RadioGroup := 100
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```

OleObject oRibbon,var_Item,var_Item1,var_Item2,var_Items

oRibbon = ole_1.Object
oRibbon.AllowToggleRadio = true
var_Items =oRibbon.Items

```

```
var_Item = var_Items.Add("Radio 1",0,1000)
    var_Item.Radio = true
    var_Item.RadioGroup = 100
var_Item1 = var_Items.Add("Radio 2",0,1001)
    var_Item1.Radio = true
    var_Item1.RadioGroup = 100
var_Item2 = var_Items.Add("Radio 2",0,1003)
    var_Item2.Radio = true
    var_Item2.RadioGroup = 100
oRibbon.Refresh()
```

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Set **ComAllowToggleRadio** to True

Variant voltems

Get Comltems to voltems

Handle holtems

Get Create (RefClass(cComltems)) to holtems

Set pvComObject of holtems to voltems

Variant voltem

Get ComAdd of holtems "Radio 1" 0 1000 to voltem

Handle holtem

Get Create (RefClass(cComltem)) to holtem

Set pvComObject of holtem to voltem

Set ComRadio of holtem to True

Set ComRadioGroup of holtem to 100

Send Destroy to holtem

Variant voltem1

Get ComAdd of holtems "Radio 2" 0 1001 to voltem1

Handle holtem1

Get Create (RefClass(cComltem)) to holtem1

Set pvComObject of holtem1 to voltem1

Set ComRadio of holtem1 to True

Set ComRadioGroup of holtem1 to 100

```

Send Destroy to holtem1
Variant voltem2
Get ComAdd of holtems "Radio 2" 0 1003 to voltem2
Handle holtem2
Get Create (RefClass(cComItem)) to holtem2
Set pvComObject of holtem2 to voltem2
    Set ComRadio of holtem2 to True
    Set ComRadioGroup of holtem2 to 100
Send Destroy to holtem2
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem, oltem1, oltem2
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oRibbon:AllowToggleRadio := .T.
    oltems := oRibbon:Items()

```



```
oltem := oltems:Add("Radio 1",0,1000)
  oltem:Radio := .T.
  oltem:RadioGroup := 100
oltem1 := oltems:Add("Radio 2",0,1001)
  oltem1:Radio := .T.
  oltem1:RadioGroup := 100
oltem2 := oltems:Add("Radio 2",0,1003)
  oltem2:Radio := .T.
  oltem2:RadioGroup := 100
oRibbon:Refresh()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
  nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
  oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Item.SelBackColor as Color

Specifies the background color of the item when it is selected.

Type	Description
Color	A Color expression that specifies the item's background color, when the item is selected/checked. 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.

The SelBackColor property specifies a different background color or a visual appearance for the item, when item is selected / checked. The [BackColor](#) property specifies the item's background color of the item. The [SelHotBackColor](#) property specifies a different background color or a visual appearance for the item, when item is selected / checked, and the cursor hovers it. The [HotBackColor](#) property specifies a different background color or a visual appearance for the item, when the cursor hovers it. The [Caption](#) property indicates the item's caption to be shown on the item. You can use the <bgcolor> HTML tag in the Caption property to specify a different background color for parts of the caption.

How can I change the item's background/backcolor, when the item is selected/checked (check-box)?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("Check 1")
      .HotBackColor = RGB(255,255,255)
      .SelBackColor = RGB(255,0,0)
      .SelHotBackColor = RGB(255,0,0)
      .ShowCheckedAsSelected = 1
      .Checked = True
      .Check = True
    End With
    .Add("").ToString = "Check 2[typ=1][chk=0][show=1][bgshot=RGB(255,255,255)]
[bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)] & _
"
```

```
End With
.Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    With .Add("Check 1")
      .HotBackColor = RGB(255,255,255)
      .SelBackColor = RGB(255,0,0)
      .SelHotBackColor = RGB(255,0,0)
      .ShowCheckedAsSelected = exDisplayItemHighlight
      .Checked = True
      .Check = True
    End With
    .Add("").ToString = "Check 2[typ=1][chk=0][show=1][bgshot=RGB(255,255,255)]
[bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]" & _
    "]"
  End With
  .Refresh
End With
```

VB.NET

```
With Exribbon1
  With .Items
    With .Add("Check 1")
      .HotBackColor = Color.FromArgb(255,255,255)
      .SelBackColor = Color.FromArgb(255,0,0)
      .SelHotBackColor = Color.FromArgb(255,0,0)
      .ShowCheckedAsSelected =
exontrol.EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight
      .Checked = True
      .Check = True
    End With
    .Add("").ToString = "Check 2[typ=1][chk=0][show=1][bgshot=RGB(255,255,255)]
[bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]" & _

```

```
""]
```

```
End With
```

```
.Refresh()
```

```
End With
```

VB.NET for /COM

```
With AxRibbon1
```

```
With .Items
```

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

```
.HotBackColor = RGB(255,255,255)
```

```
.SelBackColor = RGB(255,0,0)
```

```
.SelHotBackColor = RGB(255,0,0)
```

```
.ShowCheckedAsSelected =
```

```
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight
```

```
.Checked = True
```

```
.Check = True
```

```
End With
```

```
.Add("").ToString = "Check 2[typ=1][chk=0][show=1][bghot=RGB(255,255,255)]  
[bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]" & _
```

```
"]
```

```
End With
```

```
.Refresh()
```

```
End With
```

C++

```
/*
```

```
Copy and paste the following directives to your header file as  
it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control  
Library'
```

```
#import <ExRibbon.dll>
```

```
using namespace EXRIBBONLib;
```

```
*/
```

```
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
```

```
>GetControlUnknown();
```

```
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
```

```

EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Check 1",vtMissing,vtMissing);
var_Item->PutHotBackColor(RGB(255,255,255));
var_Item->PutSelBackColor(RGB(255,0,0));
var_Item->PutSelHotBackColor(RGB(255,0,0));
var_Item-
> PutShowCheckedAsSelected(EXRIBBONLib::exDisplayItemHighlight);
var_Item->PutChecked(VARIANT_TRUE);
var_Item->PutCheck(VARIANT_TRUE);
var_Items->Add(L"",vtMissing,vtMissing)->PutToString(_bstr_t("Check 2[typ=1]
[chk=0][show=1][bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)]
[bgsselhot=RGB(255,0,0)]" +
"");
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
Exribbonlib_tlb::IItemPtr var_Item = var_Items->Add(L"Check
1",TNoParam(),TNoParam());
var_Item->HotBackColor = RGB(255,255,255);
var_Item->SelBackColor = RGB(255,0,0);
var_Item->SelHotBackColor = RGB(255,0,0);
var_Item->ShowCheckedAsSelected =
Exribbonlib_tlb::ShowCheckedAsSelectedEnum::exDisplayItemHighlight;
var_Item->Checked = true;
var_Item->Check = true;
var_Items->Add(L"",TNoParam(),TNoParam())->ToString = TVariant(String("Check
2[typ=1][chk=0][show=1][bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)]
[bgsselhot=RGB(255,0,0)]" +
"");
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Check 1",null,null);

```

```

var_Item.HotBackColor = Color.FromArgb(255,255,255);
var_Item.SelBackColor = Color.FromArgb(255,0,0);
var_Item.SelHotBackColor = Color.FromArgb(255,0,0);
var_Item.ShowCheckedAsSelected =
exontrol.EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
var_Item.Checked = true;
var_Item.Check = true;
var_Items.Add("",null,null).ToString = "Check 2[typ=1][chk=0][show=1]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)" +
"]";
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var var_Item = var_Items.Add("Check 1",null,null);
    var_Item.HotBackColor = 16777215;
    var_Item.SelBackColor = 255;
    var_Item.SelHotBackColor = 255;
    var_Item.ShowCheckedAsSelected = 1;
    var_Item.Checked = true;
    var_Item.Check = true;
    var_Items.Add("",null,null).ToString = "Check 2[typ=1][chk=0][show=1]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)" +
"]";
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Ribbon1
    With .Items
      With .Add("Check 1")
        .HotBackColor = RGB(255,255,255)
        .SelBackColor = RGB(255,0,0)
        .SelHotBackColor = RGB(255,0,0)
        .ShowCheckedAsSelected = 1
        .Checked = True
        .Check = True
      End With
      .Add("").ToString = "Check 2[typ=1][chk=0][show=1]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)" & _
        "]"
    End With
    .Refresh
  End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;
EXRIBBONLib.Item var_Item = var_Items.Add("Check 1",null,null);
var_Item.HotBackColor =
(uint)ColorTranslator.ToWin32(Color.FromArgb(255,255,255));
var_Item.SelBackColor =
```

```

(uint)ColorTranslator.ToWin32(Color.FromArgb(255,0,0));
    var_Item.SelHotBackColor =
(uint)ColorTranslator.ToWin32(Color.FromArgb(255,0,0));
    var_Item.ShowCheckedAsSelected =
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
    var_Item.Checked = true;
    var_Item.Check = true;
    var_Items.Add("",null,null).ToString = "Check 2[typ=1][chk=0][show=1]
[bgshot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)" +
    "];
    axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Item1,com_Items;
    anytype var_Item,var_Item1,var_Items;
    str var_s;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("Check 1"); com_Item = var_Item;
    com_Item.HotBackColor(WinApi::RGB2int(255,255,255));
    com_Item.SelBackColor(WinApi::RGB2int(255,0,0));
    com_Item.SelHotBackColor(WinApi::RGB2int(255,0,0));
    com_Item.ShowCheckedAsSelected(1/*exDisplayItemHighlight*/);
    com_Item.Checked(true);
    com_Item.Check(true);
    var_Item1 = COM::createFromObject(com_Items.Add("")); com_Item1 =
var_Item1;
    var_s = "Check 2[typ=1][chk=0][show=1][bgshot=RGB(255,255,255)]
[bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)];
    com_Item1.ToString(var_s);

```



```
exribbon1.Refresh();  
}
```

Delphi 8 (.NET only)

```
with AxRibbon1 do  
begin  
  with Items do  
  begin  
    with Add('Check 1',Nil,Nil) do  
    begin  
      HotBackColor := $ffffff;  
      SelBackColor := $ff;  
      SelHotBackColor := $ff;  
      ShowCheckedAsSelected :=  
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;  
      Checked := True;  
      Check := True;  
    end;  
    Add(',Nil,Nil).ToString := 'Check 2[typ=1][chk=0][show=1]  
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]' +  
    '';  
  end;  
  Refresh();  
end
```

Delphi (standard)

```
with Ribbon1 do  
begin  
  with Items do  
  begin  
    with Add('Check 1',Null,Null) do  
    begin  
      HotBackColor := $ffffff;  
      SelBackColor := $ff;  
      SelHotBackColor := $ff;  
      ShowCheckedAsSelected := EXRIBBONLib_TLB.exDisplayItemHighlight;
```

```

        Checked := True;
        Check := True;
    end;
    Add(',Null,Null).ToString := 'Check 2[typ=1][chk=0][show=1]
[bgshot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]' +
    ";
    end;
    Refresh();
end

```

VFP

```

with thisform.Ribbon1
with .Items
    with .Add("Check 1")
        .HotBackColor = RGB(255,255,255)
        .SelBackColor = RGB(255,0,0)
        .SelHotBackColor = RGB(255,0,0)
        .ShowCheckedAsSelected = 1
        .Checked = .T.
        .Check = .T.
    endwith
    var_s = "Check 2[typ=1][chk=0][show=1][bgshot=RGB(255,255,255)]
[bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"
    .Add("").ToString = var_s
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Items

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("Check 1")
    var_Item.HotBackColor = 0xffffffff
    var_Item.SelBackColor = 0xff

```

```

var_Item.SelHotBackColor = 0xff
var_Item.ShowCheckedAsSelected = 1
var_Item.Checked = true
var_Item.Check = true
// var_Items.Add("").ToString = "Check 2[typ=1][chk=0][show=1]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"
var_Item1 = var_Items.Add("")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.ToString = "Check 2[typ=1][chk=0][show=1]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Item = var_Items.Add("Check 1")
var_Item.HotBackColor = 16777215
var_Item.SelBackColor = 255
var_Item.SelHotBackColor = 255
var_Item.ShowCheckedAsSelected = 1
var_Item.Checked = .t.
var_Item.Check = .t.
' var_Items.Add("").ToString = "Check 2[typ=1][chk=0][show=1]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"
var_Item1 = var_Items.Add("")
oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1

```

```
oRibbon.Template = "var_Item1.ToString = \"Check 2[typ=1][chk=0][show=1]  
[bgshot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]\""  
  
oRibbon.Refresh()
```

Visual Objects

```
local var_Item as IItem  
local var_Items as IItems  
  
var_Items := oDCOCX_Exontrol1.Items  
var_Item := var_Items.Add("Check 1",nil,nil)  
var_Item.HotBackColor := RGB(255,255,255)  
var_Item.SelBackColor := RGB(255,0,0)  
var_Item.SelHotBackColor := RGB(255,0,0)  
var_Item.ShowCheckedAsSelected := exDisplayItemHighlight  
var_Item.Checked := true  
var_Item.Check := true  
var_Items.Add("",nil,nil).ToString := "Check 2[typ=1][chk=0][show=1]  
[bgshot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"  
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Items  
  
oRibbon = ole_1.Object  
var_Items = oRibbon.Items  
var_Item = var_Items.Add("Check 1")  
var_Item.HotBackColor = RGB(255,255,255)  
var_Item.SelBackColor = RGB(255,0,0)  
var_Item.SelHotBackColor = RGB(255,0,0)  
var_Item.ShowCheckedAsSelected = 1  
var_Item.Checked = true  
var_Item.Check = true  
var_Items.Add("").ToString = "Check 2[typ=1][chk=0][show=1]"
```

```
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"  
oRibbon.Refresh()
```

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Variant voltems

Get Comltems to voltems

Handle holtems

Get Create (RefClass(cComltems)) to holtems

Set pvComObject of holtems to voltems

Variant voltem

Get ComAdd of holtems "Check 1" Nothing Nothing to voltem

Handle holtem

Get Create (RefClass(cComltem)) to holtem

Set pvComObject of holtem to voltem

Set **ComHotBackColor** of holtem to (RGB(255,255,255))

Set **ComSelBackColor** of holtem to (RGB(255,0,0))

Set **ComSelHotBackColor** of holtem to (RGB(255,0,0))

Set **ComShowCheckedAsSelected** of holtem to OLExDisplayItemHighlight

Set ComChecked of holtem to True

Set **ComCheck** of holtem to True

Send Destroy to holtem

Variant voltem1

Get ComAdd of holtems "" Nothing Nothing to voltem1

Handle holtem1

Get Create (RefClass(cComltem)) to holtem1

Set pvComObject of holtem1 to voltem1

Set ComToString of holtem1 to "Check 2[typ=1][chk=0][show=1]

```
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"
```

Send Destroy to holtem1

Send Destroy to holtems

Send ComRefresh

End_Procedure

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
    LOCAL oForm
```

```
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
    LOCAL oltem
```

```
    LOCAL oltems
```

```
    LOCAL oRibbon
```

```
    oForm := XbpDialog():new( AppDesktop() )
```

```
    oForm:drawingArea:clipChildren := .T.
```

```
    oForm:create( ,, {100,100}, {640,480},,, .F. )
```

```
    oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
```

```
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-  
CFBE431702E2}*/
```

```
    oRibbon:create(,, {10,60},{610,370} )
```

```
    oltems := oRibbon:Items()
```

```
        oltem := oltems:Add("Check 1")
```

```
            oltem:SetProperty("HotBackColor",AutomationTranslateColor(  
GraMakeRGBColor ( { 255,255,255 } ) , .F. ))
```

```
            oltem:SetProperty("SelBackColor",AutomationTranslateColor(  
GraMakeRGBColor ( { 255,0,0 } ) , .F. ))
```

```
            oltem:SetProperty("SelHotBackColor",AutomationTranslateColor(  
GraMakeRGBColor ( { 255,0,0 } ) , .F. ))
```

```
            oltem:ShowCheckedAsSelected := 1/*exDisplayItemHighlight*/
```

```
            oltem:Checked := .T.
```

```
            oltem:Check := .T.
```

```
            oltems:Add(""):ToString := "Check 2[typ=1][chk=0][show=1]  
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"
```

```
    oRibbon:Refresh()
```

```
    oForm:Show()
```

```
    DO WHILE nEvent != xbeP_Quit
```

```
nEvent := AppEvent( @mp1, @mp2, @oXbp )  
oXbp:handleEvent( nEvent, mp1, mp2 )  
ENDDO  
RETURN
```

property Item.SelHotBackColor as Color

Specifies the background color of the selected item when the cursor hovers it.

Type	Description
Color	A Color expression that specifies the item's background color, when the item is selected/checked and the cursor hovers it. 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.

The SelHotBackColor property specifies a different background color or a visual appearance for the item, when item is selected / checked, and the cursor hovers it. The [SelBackColor](#) property specifies a different background color or a visual appearance for the item, when item is selected / checked. The [BackColor](#) property specifies the item's background color of the item. The [HotBackColor](#) property specifies a different background color or a visual appearance for the item, when the cursor hovers it. The [Caption](#) property indicates the item's caption to be shown on the item. You can use the <bgcolor> HTML tag in the Caption property to specify a different background color for parts of the caption.

How can I change the item's background/backcolor, when the item is selected/checked (radio-buttons)?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("",2)
      .GroupPopup = 3 ' GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
    With .Items
      With .Add("Radio 1")
        .HotBackColor = RGB(255,255,255)
        .SelBackColor = RGB(255,0,0)
        .SelHotBackColor = RGB(255,0,0)
        .ShowCheckedAsSelected = 1
        .Radio = True
```



```

        .RadioGroup = 100
        .Checked = True
    End With
    With .Add("Radio 2")
        .HotBackColor = RGB(255,255,255)
        .SelBackColor = RGB(255,0,0)
        .SelHotBackColor = RGB(255,0,0)
        .ShowCheckedAsSelected = 1
        .Radio = True
        .RadioGroup = 100
    End With
    .Add("").ToString = "Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0," & _
"0)]"
    End With
End With
End With
End With
.Refresh
End With

```

VB6

```

With Ribbon1
    With .Items
        With .Add("",2)
            .GroupPopup = GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
        End With
        With .Items
            With .Add("Radio 1")
                .HotBackColor = RGB(255,255,255)
                .SelBackColor = RGB(255,0,0)
                .SelHotBackColor = RGB(255,0,0)
                .ShowCheckedAsSelected = exDisplayItemHighlight
                .Radio = True
                .RadioGroup = 100
                .Checked = True
            End With
        End With
    End With
End With

```

```

With .Add("Radio 2")
    .HotBackColor = RGB(255,255,255)
    .SelBackColor = RGB(255,0,0)
    .SelHotBackColor = RGB(255,0,0)
    .ShowCheckedAsSelected = exDisplayItemHighlight
    .Radio = True
    .RadioGroup = 100
End With
.Add("").ToString = "Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0," & _
"0)]"
End With
End With
End With
.Refresh
End With

```

VB.NET

```

With Exribbon1
    With .Items
        With .Add("",2)
            .GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup
        With .Items
            With .Add("Radio 1")
                .HotBackColor = Color.FromArgb(255,255,255)
                .SelBackColor = Color.FromArgb(255,0,0)
                .SelHotBackColor = Color.FromArgb(255,0,0)
                .ShowCheckedAsSelected =
exontrol.EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight
                .Radio = True
                .RadioGroup = 100
                .Checked = True
            End With
            With .Add("Radio 2")

```

```

        .HotBackColor = Color.FromArgb(255,255,255)
        .SelBackColor = Color.FromArgb(255,0,0)
        .SelHotBackColor = Color.FromArgb(255,0,0)
        .ShowCheckedAsSelected =
exontrol.EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight
        .Radio = True
        .RadioGroup = 100
    End With
        .Add("").ToString = "Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0," & _
"0)]"
    End With
End With
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    With .Items
        With .Add("",2)
            .GroupPopup = EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or
EXRIBBONLib.GroupPopupEnum.exGroupPopup
        With .Items
            With .Add("Radio 1")
                .HotBackColor = RGB(255,255,255)
                .SelBackColor = RGB(255,0,0)
                .SelHotBackColor = RGB(255,0,0)
                .ShowCheckedAsSelected =
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight
                .Radio = True
                .RadioGroup = 100
                .Checked = True
            End With
            With .Add("Radio 2")
                .HotBackColor = RGB(255,255,255)

```

```

        .SelBackColor = RGB(255,0,0)
        .SelHotBackColor = RGB(255,0,0)
        .ShowCheckedAsSelected =
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight
        .Radio = True
        .RadioGroup = 100
    End With
        .Add("").ToString = "Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0," & _
"0)]"
    End With
End With
End With
End With
.Refresh()
End With

```

C++

```

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"",long(2),vtMissing);
    var_Item-
>PutGroupPopup(EXRIBBONLib::GroupPopupEnum(EXRIBBONLib::exNoGroupPopupF
| EXRIBBONLib::exGroupPopup));
    EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();
    EXRIBBONLib::IItemPtr var_Item1 = var_Items1->Add(L"Radio
1",vtMissing,vtMissing);
    var_Item1->PutHotBackColor(RGB(255,255,255));

```

```

var_Item1->PutSelBackColor(RGB(255,0,0));
var_Item1->PutSelHotBackColor(RGB(255,0,0));
var_Item1->
> PutShowCheckedAsSelected(EXRIBBONLib::exDisplayItemHighlight);
var_Item1->PutRadio(VARIANT_TRUE);
var_Item1->PutRadioGroup(100);
var_Item1->PutChecked(VARIANT_TRUE);
EXRIBBONLib::IItemPtr var_Item2 = var_Items1->Add(L"Radio
2",vtMissing,vtMissing);
var_Item2->PutHotBackColor(RGB(255,255,255));
var_Item2->PutSelBackColor(RGB(255,0,0));
var_Item2->PutSelHotBackColor(RGB(255,0,0));
var_Item2->
> PutShowCheckedAsSelected(EXRIBBONLib::exDisplayItemHighlight);
var_Item2->PutRadio(VARIANT_TRUE);
var_Item2->PutRadioGroup(100);
var_Items1->Add(L"",vtMissing,vtMissing)->PutToString(_bstr_t("Radio
3[typ=2][show=1][grp=100][bgshot=RGB(255,255,255)][bgssel=RGB(255,0,0)]
[bgsselhot=RGB(255,0,") +
"0]");
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
Exribbonlib_tlb::IItemPtr var_Item = var_Items->Add(L"",TVariant(2),TNoParam());
var_Item->GroupPopup =
Exribbonlib_tlb::GroupPopupEnum::exNoGroupPopupFrame |
Exribbonlib_tlb::GroupPopupEnum::exGroupPopup;
Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;
Exribbonlib_tlb::IItemPtr var_Item1 = var_Items1->Add(L"Radio
1",TNoParam(),TNoParam());
var_Item1->HotBackColor = RGB(255,255,255);
var_Item1->SelBackColor = RGB(255,0,0);
var_Item1->SelHotBackColor = RGB(255,0,0);
var_Item1->ShowCheckedAsSelected =

```

```

Exribbonlib_tlb::ShowCheckedAsSelectedEnum::exDisplayItemHighlight;
    var_Item1->Radio = true;
    var_Item1->RadioGroup = 100;
    var_Item1->Checked = true;
    Exribbonlib_tlb::IItemPtr var_Item2 = var_Items1->Add(L"Radio
2",TNoParam(),TNoParam());
    var_Item2->HotBackColor = RGB(255,255,255);
    var_Item2->SelBackColor = RGB(255,0,0);
    var_Item2->SelHotBackColor = RGB(255,0,0);
    var_Item2->ShowCheckedAsSelected =
Exribbonlib_tlb::ShowCheckedAsSelectedEnum::exDisplayItemHighlight;
    var_Item2->Radio = true;
    var_Item2->RadioGroup = 100;
    var_Items1->Add(L"",TNoParam(),TNoParam())->ToString =
TVariant(String("Radio 3[typ=2][show=1][grp=100][bgshot=RGB(255,255,255)]
[bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]");
    Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
    exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
    var_Item.GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup;
    exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
    exontrol.EXRIBBONLib.Item var_Item1 = var_Items1.Add("Radio 1",null,null);
    var_Item1.HotBackColor = Color.FromArgb(255,255,255);
    var_Item1.SelBackColor = Color.FromArgb(255,0,0);
    var_Item1.SelHotBackColor = Color.FromArgb(255,0,0);
    var_Item1.ShowCheckedAsSelected =
exontrol.EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
    var_Item1.Radio = true;
    var_Item1.RadioGroup = 100;
    var_Item1.Checked = true;

```

```

exontrol.EXRIBBONLib.Item var_Item2 = var_Items1.Add("Radio 2",null,null);
var_Item2.HotBackColor = Color.FromArgb(255,255,255);
var_Item2.SelBackColor = Color.FromArgb(255,0,0);
var_Item2.SelHotBackColor = Color.FromArgb(255,0,0);
var_Item2.ShowCheckedAsSelected =
exontrol.EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
var_Item2.Radio = true;
var_Item2.RadioGroup = 100;
var_Items1.Add("",null,null).ToString = "Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0," +
"0)"]";
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
var var_Items = Ribbon1.Items;
var var_Item = var_Items.Add("",2,null);
var_Item.GroupPopup = 3;
var var_Items1 = var_Item.Items;
var var_Item1 = var_Items1.Add("Radio 1",null,null);
var_Item1.HotBackColor = 16777215;
var_Item1.SelBackColor = 255;
var_Item1.SelHotBackColor = 255;
var_Item1.ShowCheckedAsSelected = 1;
var_Item1.Radio = true;
var_Item1.RadioGroup = 100;
var_Item1.Checked = true;
var var_Item2 = var_Items1.Add("Radio 2",null,null);
var_Item2.HotBackColor = 16777215;

```

```

var_Item2.SelBackColor = 255;
var_Item2.SelHotBackColor = 255;
var_Item2.ShowCheckedAsSelected = 1;
var_Item2.Radio = true;
var_Item2.RadioGroup = 100;
var_Items1.Add("",null,null).ToString = "Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0," +
"0)"]";
Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            With .Add("",2)
                .GroupPopup = 3 ' GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
            With .Items
                With .Add("Radio 1")
                    .HotBackColor = RGB(255,255,255)
                    .SelBackColor = RGB(255,0,0)
                    .SelHotBackColor = RGB(255,0,0)
                    .ShowCheckedAsSelected = 1
                    .Radio = True
                    .RadioGroup = 100
                    .Checked = True
                End With
            End With
        End With
    End With
End Function

```



```

        With .Add("Radio 2")
            .HotBackColor = RGB(255,255,255)
            .SelBackColor = RGB(255,0,0)
            .SelHotBackColor = RGB(255,0,0)
            .ShowCheckedAsSelected = 1
            .Radio = True
            .RadioGroup = 100
        End With
        .Add("").ToString = "Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0," & _
"0)]"
        End With
    End With
End With
.Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
    EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
    var_Item.GroupPopup =
EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
EXRIBBONLib.GroupPopupEnum.exGroupPopup;
    EXRIBBONLib.Items var_Items1 = var_Item.Items;
    EXRIBBONLib.Item var_Item1 = var_Items1.Add("Radio 1",null,null);
    var_Item1.HotBackColor =
(uint)ColorTranslator.ToWin32(Color.FromArgb(255,255,255));
    var_Item1.SelBackColor =
(uint)ColorTranslator.ToWin32(Color.FromArgb(255,0,0));
    var_Item1.SelHotBackColor =
(uint)ColorTranslator.ToWin32(Color.FromArgb(255,0,0));
    var_Item1.ShowCheckedAsSelected =

```

```

EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
    var_Item1.Radio = true;
    var_Item1.RadioGroup = 100;
    var_Item1.Checked = true;
    EXRIBBONLib.Item var_Item2 = var_Items1.Add("Radio 2",null,null);
    var_Item2.HotBackColor =
(uint)ColorTranslator.ToWin32(Color.FromArgb(255,255,255));
    var_Item2.SelBackColor =
(uint)ColorTranslator.ToWin32(Color.FromArgb(255,0,0));
    var_Item2.SelHotBackColor =
(uint)ColorTranslator.ToWin32(Color.FromArgb(255,0,0));
    var_Item2.ShowCheckedAsSelected =
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
    var_Item2.Radio = true;
    var_Item2.RadioGroup = 100;
    var_Items1.Add("",null,null).ToString = "Radio 3[typ=2][show=1][grp=100]
[bgshot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0," +
"0)"]";
    axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Item1,com_Item2,com_Item3,com_Items,com_Items1;
    anytype var_Item,var_Item1,var_Item2,var_Item3,var_Items,var_Items1;
    str var_s;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("",COMVariant::createFromInt(2)); com_Item =
var_Item;
    com_Item.GroupPopup(3/*exNoGroupPopupFrame | exGroupPopup*/);
    var_Items1 = com_Item.Items(); com_Items1 = var_Items1;

```

```

var_Item1 = com_Items1.Add("Radio 1"); com_Item1 = var_Item1;
  com_Item1.HotBackColor(WinApi::RGB2int(255,255,255));
  com_Item1.SelBackColor(WinApi::RGB2int(255,0,0));
  com_Item1.SelHotBackColor(WinApi::RGB2int(255,0,0));
  com_Item1.ShowCheckedAsSelected(1/*exDisplayItemHighlight*/);
  com_Item1.Radio(true);
  com_Item1.RadioGroup(100);
  com_Item1.Checked(true);
var_Item2 = com_Items1.Add("Radio 2"); com_Item2 = var_Item2;
  com_Item2.HotBackColor(WinApi::RGB2int(255,255,255));
  com_Item2.SelBackColor(WinApi::RGB2int(255,0,0));
  com_Item2.SelHotBackColor(WinApi::RGB2int(255,0,0));
  com_Item2.ShowCheckedAsSelected(1/*exDisplayItemHighlight*/);
  com_Item2.Radio(true);
  com_Item2.RadioGroup(100);
var_Item3 = COM::createFromObject(com_Items1.Add("")); com_Item3 =
var_Item3;
  var_s = "Radio 3[typ=2][show=1][grp=100][bgshot=RGB(255,255,255)]
[bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)";
  var_s = var_s + "]]";
  com_Item3.ToString(var_s);
exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
  with Items do
  begin
    with Add('',TObject(2),Nil) do
    begin
      GroupPopup :=
Integer(EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib.GroupPopupEnum.exGroupPopup);
      with Items do
      begin

```

```

with Add('Radio 1',Nil,Nil) do
begin
  HotBackColor := $ffffff;
  SelBackColor := $ff;
  SelHotBackColor := $ff;
  ShowCheckedAsSelected :=
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
  Radio := True;
  RadioGroup := 100;
  Checked := True;
end;
with Add('Radio 2',Nil,Nil) do
begin
  HotBackColor := $ffffff;
  SelBackColor := $ff;
  SelHotBackColor := $ff;
  ShowCheckedAsSelected :=
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
  Radio := True;
  RadioGroup := 100;
end;
Add('',Nil,Nil).ToString := 'Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0' +
  ')]';
end;
end;
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  with Items do
  begin
    with Add('',OleVariant(2),Null) do

```

```

begin
  GroupPopup := Integer(EXRIBBONLib_TLB.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib_TLB.exGroupPopup);
  with Items do
    begin
      with Add('Radio 1',Null,Null) do
        begin
          HotBackColor := $ffffff;
          SelBackColor := $ff;
          SelHotBackColor := $ff;
          ShowCheckedAsSelected := EXRIBBONLib_TLB.exDisplayItemHighlight;
          Radio := True;
          RadioGroup := 100;
          Checked := True;
        end;
      with Add('Radio 2',Null,Null) do
        begin
          HotBackColor := $ffffff;
          SelBackColor := $ff;
          SelHotBackColor := $ff;
          ShowCheckedAsSelected := EXRIBBONLib_TLB.exDisplayItemHighlight;
          Radio := True;
          RadioGroup := 100;
        end;
      Add(',Null,Null).ToString := 'Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0' +
      ')';
    end;
  end;
end;
Refresh();
end

```

VFP

```

with thisform.Ribbon1
  with .Items

```

```

with .Add("",2)
    .GroupPopup = 3 && GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
with .Items
    with .Add("Radio 1")
        .HotBackColor = RGB(255,255,255)
        .SelBackColor = RGB(255,0,0)
        .SelHotBackColor = RGB(255,0,0)
        .ShowCheckedAsSelected = 1
        .Radio = .T.
        .RadioGroup = 100
        .Checked = .T.
    endwith
    with .Add("Radio 2")
        .HotBackColor = RGB(255,255,255)
        .SelBackColor = RGB(255,0,0)
        .SelHotBackColor = RGB(255,0,0)
        .ShowCheckedAsSelected = 1
        .Radio = .T.
        .RadioGroup = 100
    endwith
    var_s = "Radio 3[typ=2][show=1][grp=100][bgshot=RGB(255,255,255)]
[bgsel=RGB(255,0,0)][bgselhot=RGB(255,0,0)"
    var_s = var_s + "]"
    .Add("").ToString = var_s
endwith
endwith
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Item2,var_Item3,var_Items,var_Items1

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items

```

```

var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
var_Items1 = var_Item.Items
var_Item1 = var_Items1.Add("Radio 1")
var_Item1.HotBackColor = 0xffffffff
var_Item1.SelBackColor = 0xff
var_Item1.SelHotBackColor = 0xff
var_Item1.ShowCheckedAsSelected = 1
var_Item1.Radio = true
var_Item1.RadioGroup = 100
var_Item1.Checked = true
var_Item2 = var_Items1.Add("Radio 2")
var_Item2.HotBackColor = 0xffffffff
var_Item2.SelBackColor = 0xff
var_Item2.SelHotBackColor = 0xff
var_Item2.ShowCheckedAsSelected = 1
var_Item2.Radio = true
var_Item2.RadioGroup = 100
// var_Items1.Add("").ToString = "Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"
var_Item3 = var_Items1.Add("")
with (oRibbon)
    TemplateDef = [Dim var_Item3]
    TemplateDef = var_Item3
    Template = [var_Item3.ToString = "Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Item2 as P
Dim var_Item3 as P

```

Dim var_Items as P

Dim var_Items1 as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex

var_Items = oRibbon.Items

var_Item = var_Items.Add("",2)

var_Item.GroupPopup = 3 'exNoGroupPopupFrame + exGroupPopup

var_Items1 = var_Item.Items

var_Item1 = var_Items1.Add("Radio 1")

var_Item1.HotBackColor = 16777215

var_Item1.SelBackColor = 255

var_Item1.SelHotBackColor = 255

var_Item1.ShowCheckedAsSelected = 1

var_Item1.Radio = .t.

var_Item1.RadioGroup = 100

var_Item1.Checked = .t.

var_Item2 = var_Items1.Add("Radio 2")

var_Item2.HotBackColor = 16777215

var_Item2.SelBackColor = 255

var_Item2.SelHotBackColor = 255

var_Item2.ShowCheckedAsSelected = 1

var_Item2.Radio = .t.

var_Item2.RadioGroup = 100

' var_Items1.Add("").ToString = "Radio 3[typ=2][show=1][grp=100]
[bgshot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"

var_Item3 = var_Items1.Add("")

oRibbon.TemplateDef = "Dim var_Item3"

oRibbon.TemplateDef = var_Item3

oRibbon.Template = "var_Item3.ToString = \"Radio 3[typ=2][show=1]
[grp=100][bgshot=RGB(255,255,255)][bgssel=RGB(255,0,0)]
[bgsselhot=RGB(255,0,0)]\""

oRibbon.Refresh()


```

local var_Item,var_Item1,var_Item2 as Item
local var_Items,var_Items1 as Items

var_Items := oDCOCX_Exontrol1:Items
var_Item := var_Items:Add("",2,nil)
var_Item:GroupPopup := exNoGroupPopupFrame | exGroupPopup
var_Items1 := var_Item:Items
var_Item1 := var_Items1:Add("Radio 1",nil,nil)
var_Item1:HotBackColor := RGB(255,255,255)
var_Item1:SelBackColor := RGB(255,0,0)
var_Item1:SelHotBackColor := RGB(255,0,0)
var_Item1:ShowCheckedAsSelected := exDisplayItemHighlight
var_Item1:Radio := true
var_Item1:RadioGroup := 100
var_Item1:Checked := true
var_Item2 := var_Items1:Add("Radio 2",nil,nil)
var_Item2:HotBackColor := RGB(255,255,255)
var_Item2:SelBackColor := RGB(255,0,0)
var_Item2:SelHotBackColor := RGB(255,0,0)
var_Item2:ShowCheckedAsSelected := exDisplayItemHighlight
var_Item2:Radio := true
var_Item2:RadioGroup := 100
var_Items1:Add("",nil,nil):ToString := "Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"
oDCOCX_Exontrol1:Refresh()

```

PowerBuilder

```

OleObject oRibbon,var_Item,var_Item1,var_Item2,var_Items,var_Items1

oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
var_Items1 = var_Item.Items
var_Item1 = var_Items1.Add("Radio 1")

```

```

var_Item1.HotBackColor = RGB(255,255,255)
var_Item1.SelBackColor = RGB(255,0,0)
var_Item1.SelHotBackColor = RGB(255,0,0)
var_Item1.ShowCheckedAsSelected = 1
var_Item1.Radio = true
var_Item1.RadioGroup = 100
var_Item1.Checked = true
var_Item2 = var_Items1.Add("Radio 2")
var_Item2.HotBackColor = RGB(255,255,255)
var_Item2.SelBackColor = RGB(255,0,0)
var_Item2.SelHotBackColor = RGB(255,0,0)
var_Item2.ShowCheckedAsSelected = 1
var_Item2.Radio = true
var_Item2.RadioGroup = 100
var_Items1.Add("").ToString = "Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"
oRibbon.Refresh()

```

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Variant voltems

Get Comltems to voltems

Handle holtems

Get Create (RefClass(cComltems)) to holtems

Set pvComObject of holtems to voltems

Variant voltem

Get ComAdd of holtems "" 2 Nothing to voltem

Handle holtem

Get Create (RefClass(cComltem)) to holtem

Set pvComObject of holtem to voltem

Set ComGroupPopup of holtem to (OLEexNoGroupPopupFrame +
OLEexGroupPopup)

Variant voltems1

Get Comltems of holtem to voltems1

Handle holtems1

Get Create (RefClass(cComltems)) to holtems1

Set pvComObject of holtems1 to voltems1

Variant voltem1

Get ComAdd of holtems1 "Radio 1" Nothing Nothing to voltem1

Handle holtem1

Get Create (RefClass(cComltem)) to holtem1

Set pvComObject of holtem1 to voltem1

Set **ComHotBackColor** of holtem1 to (RGB(255,255,255))

Set **ComSelBackColor** of holtem1 to (RGB(255,0,0))

Set **ComSelHotBackColor** of holtem1 to (RGB(255,0,0))

Set **ComShowCheckedAsSelected** of holtem1 to

OLExDisplayItemHighlight

Set ComRadio of holtem1 to True

Set ComRadioGroup of holtem1 to 100

Set ComChecked of holtem1 to True

Send Destroy to holtem1

Variant voltem2

Get ComAdd of holtems1 "Radio 2" Nothing Nothing to voltem2

Handle holtem2

Get Create (RefClass(cComltem)) to holtem2

Set pvComObject of holtem2 to voltem2

Set **ComHotBackColor** of holtem2 to (RGB(255,255,255))

Set **ComSelBackColor** of holtem2 to (RGB(255,0,0))

Set **ComSelHotBackColor** of holtem2 to (RGB(255,0,0))

Set **ComShowCheckedAsSelected** of holtem2 to

OLExDisplayItemHighlight

Set ComRadio of holtem2 to True

Set ComRadioGroup of holtem2 to 100

Send Destroy to holtem2

Variant voltem3

Get ComAdd of holtems1 "" Nothing Nothing to voltem3

Handle holtem3

Get Create (RefClass(cComltem)) to holtem3

Set pvComObject of holtem3 to voltem3

Set ComToString of holtem3 to "Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"

```
Send Destroy to holtem3
Send Destroy to holtems1
Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem,oltem1,oltem2
    LOCAL oltems,oltems1
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oltems := oRibbon:Items()
    oltem := oltems:Add("",2)
    oltem:GroupPopup := 3/*exNoGroupPopupFrame+exGroupPopup*/
    oltems1 := oltem:Items()
    oltem1 := oltems1:Add("Radio 1")
    oltem1:SetProperty("HotBackColor",AutomationTranslateColor(
GraMakeRGBColor ( { 255,255,255 } ) , .F. ))
    oltem1:SetProperty("SelBackColor",AutomationTranslateColor(
```

```

GraMakeRGBColor ( { 255,0,0 } ) , .F. ))
    oltem1:SetProperty("SelHotBackColor",AutomationTranslateColor(
GraMakeRGBColor ( { 255,0,0 } ) , .F. ))
    oltem1:ShowCheckedAsSelected := 1/*exDisplayItemHighlight*/
    oltem1:Radio := .T.
    oltem1:RadioGroup := 100
    oltem1:Checked := .T.
    oltem2 := oltems1:Add("Radio 2")
    oltem2:SetProperty("HotBackColor",AutomationTranslateColor(
GraMakeRGBColor ( { 255,255,255 } ) , .F. ))
    oltem2:SetProperty("SelBackColor",AutomationTranslateColor(
GraMakeRGBColor ( { 255,0,0 } ) , .F. ))
    oltem2:SetProperty("SelHotBackColor",AutomationTranslateColor(
GraMakeRGBColor ( { 255,0,0 } ) , .F. ))
    oltem2:ShowCheckedAsSelected := 1/*exDisplayItemHighlight*/
    oltem2:Radio := .T.
    oltem2:RadioGroup := 100
    oltems1:Add(""):ToString := "Radio 3[typ=2][show=1][grp=100]
[bghot=RGB(255,255,255)][bgssel=RGB(255,0,0)][bgsselhot=RGB(255,0,0)]"
    oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

property Item.Shortcut as String

Specifies the key combination that the user can press to select the item quickly.

Type	Description
String	A String expression that specifies the key combination the user can select the item by keys. The shortcut should include modifier keys such as ALT, CTRL and Shift

Currently, the control supports Advanced Shortcut Keys Support, including UI visual appearance. By default, the Shortcut property is empty, which indicates that no shortcut key is associated with the item. The Shortcut property specifies the key combination that the user can press to select the item quickly. The [Enabled](#) property specifies whether the item is enabled or disabled, inclusive its shortcut key. The [ShortcutKeyVisible](#) property gets or sets a value that specifies whether the control's shortcut keys are visible or hidden. The [ShortcutKeysInfo](#) property returns the list of shortcut keys that are currently available.

The Shortcut property can specify no modifier key, and so the item's shortcut became the parent's shortcut modifier plus the shortcut itself. For instance, if the parent item has the shortcut "CTRL + A", and the item's shortcut is "P", it indicates that the item's shortcut is actually "CTRL + A + P", which indicates that the item is selected as soon as the user presses the CTRL + A, and then P key. You can use this option to easily assign shortcut to items, by inheriting the parent's shortcut. At least one shortcut of any parent should have a modifier key, else the shortcut of the item itself can not be invoked.

For instance:

- "ALT + 1", specifies that the user can press ALT then 1, or together ALT and 1
- "CTRL + ALT + Enter", specifies that the user can press CTRL, then ALT and Enter or ALT , and then CTRL and Enter, or all together CTRL and ALT and Enter
- "CTRL + AA" or "CTRL + A + A", specifies that the user need to press CTRL, then A and then A

The Shortcut property supports the following modifier keys (code):

- SHIFT (16 or 0x10)
- CTRL (17 or 0x11)
- ALT (18 or 0x12)

The Shortcut property supports the following predefined keys (code):

- Add (107 or 0x6B)
- Apps (93 or 0x5D)
- Backspace (8 or 0x08)

- CapsLock (20 or 0x14)
- Comma (188 or 0xBC)
- Decimal (110 or 0x6E)
- Delete (46 or 0x2E)
- Divide (111 or 0x6F)
- Down (40 or 0x28)
- End (35 or 0x23)
- Enter (13 or 0x0D)
- Escape (27 or 0x1B)
- F1 (112 or 0x70) to F24 (135 or 0x87)
- Help (47 or 0x2F)
- Home (36 or 0x24)
- Insert (45 or 0x2D)
- LWin (91 or 0x5B)
- Left (37 or 0x25)
- Minus (189 or 0xBD)
- Multiply (106 or 0x6A)
- NumLock (144 or 0x90)
- NumPad0 (96 or 0x60) to NumPad9 (105 or 0x69)
- OEM1 (186 or 0xBA)
- OEM2 (191 or 0xBF)
- OEM3 (192 or 0xC0)
- OEM4 (219 or 0xDB)
- OEM5 (220 or 0xDC)
- OEM6 (221 or 0xDD)
- OEM7 (222 or 0xDE)
- OEM8 (223 or 0xDF)
- PageDown (34 or 0x22)
- PageUp (33 or 0x21)
- Pause (19 or 0x13)
- Period (190 or 0xBE)
- Plus (187 or 0xBB)
- PrintScr (44 or 0x2C)
- RWin (92 or 0x5C)
- Right (39 or 0x27)
- Scroll (145 or 0x91)
- Separator (108 or 0x6C)
- Sleep (95 or 0x5F)
- Space (32 or 0x20)
- Subtract (109 or 0x6D)
- Tab (9 or 0x09)
- Up (38 or 0x26)

Also, any digit (0 - 9) or letter (A - Z) is supported.

(previously) By default, the Shortcut property is defined as first sequence found in the item's [Caption](#) between <u> and </u> HTML tags. Pressing the shortcut key is similar with selecting the item and pressing the Enter key. The shortcuts in the context menu have effect only if the [PopupIncrementalSearch](#) property is exNoIncrementalSearch.

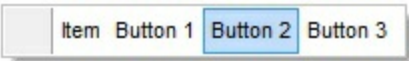
property Item.ShowAsButton as ShowAsButtonEnum

Specifies whether the item is shown as a button.

Type	Description
ShowAsButtonEnum	A ShowAsButtonEnum expression that indicates whether the item is shown as a button.

By default, the ShowAsButton property is False. Use the ShowAsButton property to add buttons to your item. The [Caption](#) property specifies the caption of the item/button. Use the Item's [CloseOnClick](#) property to specify a different way to close the menu when user clicks a specified item. You can use the ShowAsButton property on exShowAsSelectButton, for a popup-item, where the [SubMenu](#) property determines the sub-menu/items to be shown when user clicks the associated arrow (select button).

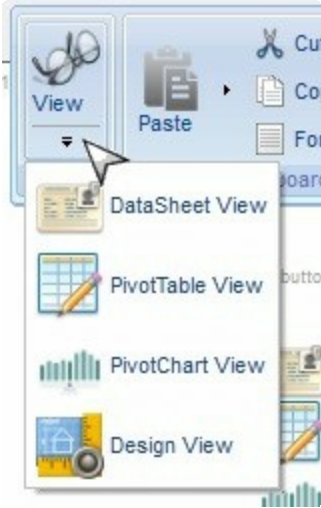
The following screen shot shows the items with no button appearance (ShowAsButton on exShowAsButtonNone)



The following screen shot shows the items with button appearance (ShowAsButton on exShowAsButton)



The following screen shot shows the items with button appearance (ShowAsButton on exShowAsSelectButtonBottom)



I am using exShowAsSelectButton/exShowAsSelectButtonBottom but none of them works. What could be wrong?

With Ribbon1

With **.Items**

With .Add("Button",2)

.ShowAsButton = 19 ' **ShowAsButtonEnum.exShowAsSelectButton Or
ShowAsButtonEnum.exShowAsButtonAutoSize**

With **.Items**

.PopupAppearance = 6

.Add "Item 1"

.Add "Item 2"

.Add "Item 3"

End With

End With

End With

.Refresh

End With

VB6

With Ribbon1

With **.Items**

With .Add("Button",2)

.ShowAsButton = ShowAsButtonEnum.exShowAsSelectButton Or
ShowAsButtonEnum.exShowAsButtonAutoSize

With **.Items**

.PopupAppearance = ShadowBorder

.Add "Item 1"

.Add "Item 2"

.Add "Item 3"

End With

End With

End With

.Refresh

End With

VB.NET

With Exribbon1

With **.Items**

```

    With .Add("Button",2)
        .ShowAsButton =
exontrol.EXRIBBONLib.ShowAsButtonEnum.exShowAsSelectButton Or
exontrol.EXRIBBONLib.ShowAsButtonEnum.exShowAsButtonAutoSize
        With .Items
            .PopupAppearance =
exontrol.EXRIBBONLib.AppearanceEnum.ShadowBorder
            .Add("Item 1")
            .Add("Item 2")
            .Add("Item 3")
        End With
    End With
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    With .Items
        With .Add("Button",2)
            .ShowAsButton = EXRIBBONLib.ShowAsButtonEnum.exShowAsSelectButton
Or EXRIBBONLib.ShowAsButtonEnum.exShowAsButtonAutoSize
            With .Items
                .PopupAppearance = EXRIBBONLib.AppearanceEnum.ShadowBorder
                .Add("Item 1")
                .Add("Item 2")
                .Add("Item 3")
            End With
        End With
    End With
End With
.Refresh()
End With

```

C++

```
/*
```

Copy and paste the following directives to your header file as

it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control Library'

```
#import <ExRibbon.dll>
using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Button",long(2),vtMissing);
    var_Item-
>PutShowAsButton(EXRIBBONLib::ShowAsButtonEnum(EXRIBBONLib::exShowAsSelect
| EXRIBBONLib::exShowAsButtonAutoSize));
    EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();
    var_Items1->PutPopupAppearance(EXRIBBONLib::ShadowBorder);
    var_Items1->Add(L"Item 1",vtMissing,vtMissing);
    var_Items1->Add(L"Item 2",vtMissing,vtMissing);
    var_Items1->Add(L"Item 3",vtMissing,vtMissing);
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    Exribbonlib_tlb::IItemPtr var_Item = var_Items-
>Add(L"Button",TVariant(2),TNoParam());
    var_Item->ShowAsButton =
Exribbonlib_tlb::ShowAsButtonEnum::exShowAsSelectButton |
Exribbonlib_tlb::ShowAsButtonEnum::exShowAsButtonAutoSize;
    Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;
    var_Items1->PopupAppearance =
Exribbonlib_tlb::AppearanceEnum::ShadowBorder;
    var_Items1->Add(L"Item 1",TNoParam(),TNoParam());
    var_Items1->Add(L"Item 2",TNoParam(),TNoParam());
    var_Items1->Add(L"Item 3",TNoParam(),TNoParam());
Ribbon1->Refresh();
```

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
    exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Button",2,null);
    var_Item.ShowAsButton =
exontrol.EXRIBBONLib.ShowAsButtonEnum.exShowAsSelectButton |
exontrol.EXRIBBONLib.ShowAsButtonEnum.exShowAsButtonAutoSize;
    exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
    var_Items1.PopupAppearance =
exontrol.EXRIBBONLib.AppearanceEnum.ShadowBorder;
    var_Items1.Add("Item 1",null,null);
    var_Items1.Add("Item 2",null,null);
    var_Items1.Add("Item 3",null,null);
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var var_Item = var_Items.Add("Button",2,null);
    var_Item.ShowAsButton = 19;
    var var_Items1 = var_Item.Items;
    var_Items1.PopupAppearance = 6;
    var_Items1.Add("Item 1",null,null);
    var_Items1.Add("Item 2",null,null);
    var_Items1.Add("Item 3",null,null);
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            With .Add("Button",2)
                .ShowAsButton = 19 ' ShowAsButtonEnum.exShowAsSelectButton Or
ShowAsButtonEnum.exShowAsButtonAutoSize
            With .Items
                .PopupAppearance = 6
                .Add "Item 1"
                .Add "Item 2"
                .Add "Item 3"
            End With
        End With
    End With
    .Refresh
End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;
EXRIBBONLib.Item var_Item = var_Items.Add("Button",2,null);
var_Item.ShowAsButton =
EXRIBBONLib.ShowAsButtonEnum.exShowAsSelectButton |
EXRIBBONLib.ShowAsButtonEnum.exShowAsButtonAutoSize;
EXRIBBONLib.Items var_Items1 = var_Item.Items;
var_Items1.PopupAppearance =
EXRIBBONLib.AppearanceEnum.ShadowBorder;
```

```

var_Items1.Add("Item 1",null,null);
var_Items1.Add("Item 2",null,null);
var_Items1.Add("Item 3",null,null);
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Items,com_Items1;
    anytype var_Item,var_Items,var_Items1;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("Button",COMVariant::createFromInt(2)); com_Item =
var_Item;
    com_Item.ShowAsButton(19/*exShowAsSelectButton |
exShowAsButtonAutoSize*/);
    var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
    com_Items1.PopupAppearance(6/*ShadowBorder*/);
    com_Items1.Add("Item 1");
    com_Items1.Add("Item 2");
    com_Items1.Add("Item 3");
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        with Add('Button',TObject(2),Nil) do
        begin
            ShowAsButton :=

```

```

Integer(EXRIBBONLib.ShowAsButtonEnum.exShowAsSelectButton) Or
Integer(EXRIBBONLib.ShowAsButtonEnum.exShowAsButtonAutoSize);
  with Items do
  begin
    PopupAppearance := EXRIBBONLib.AppearanceEnum.ShadowBorder;
    Add('Item 1',Nil,Nil);
    Add('Item 2',Nil,Nil);
    Add('Item 3',Nil,Nil);
  end;
end;
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  with Items do
  begin
    with Add('Button',OleVariant(2),Null) do
    begin
      ShowAsButton := Integer(EXRIBBONLib_TLB.exShowAsSelectButton) Or
Integer(EXRIBBONLib_TLB.exShowAsButtonAutoSize);
      with Items do
      begin
        PopupAppearance := EXRIBBONLib_TLB.ShadowBorder;
        Add('Item 1',Null,Null);
        Add('Item 2',Null,Null);
        Add('Item 3',Null,Null);
      end;
    end;
  end;
end;
Refresh();
end

```



```

with thisform.Ribbon1
  with .Items
    with .Add("Button",2)
      .ShowAsButton = 19 && ShowAsButtonEnum.exShowAsSelectButton Or
ShowAsButtonEnum.exShowAsButtonAutoSize
    with .Items
      .PopupAppearance = 6
      .Add("Item 1")
      .Add("Item 2")
      .Add("Item 3")
    endwith
  endwith
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Items,var_Items1

oRibbon = form.Active1.nativeObject
var_Items = oRibbon.Items
  var_Item = var_Items.Add("Button",2)
  var_Item.ShowAsButton = 19 /*exShowAsSelectButton |
exShowAsButtonAutoSize*/
  var_Items1 = var_Item.Items
  var_Items1.PopupAppearance = 6
  var_Items1.Add("Item 1")
  var_Items1.Add("Item 2")
  var_Items1.Add("Item 3")
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Items as P

```

```
Dim var_Items1 as P
```

```
oRibbon = topparent:CONTROL_ACTIVEX1.activex
```

```
var_Items = oRibbon.Items
```

```
var_Item = var_Items.Add("Button",2)
```

```
var_Item.ShowAsButton = 19 'exShowAsSelectButton +  
exShowAsButtonAutoSize
```

```
var_Items1 = var_Item.Items
```

```
var_Items1.PopupAppearance = 6
```

```
var_Items1.Add("Item 1")
```

```
var_Items1.Add("Item 2")
```

```
var_Items1.Add("Item 3")
```

```
oRibbon.Refresh()
```

Visual Objects

```
local var_Item as IItem
```

```
local var_Items,var_Items1 as IItems
```

```
var_Items := oDCOCX_Exontrol1:Items
```

```
var_Item := var_Items.Add("Button",2,nil)
```

```
var_Item.ShowAsButton := exShowAsSelectButton | exShowAsButtonAutoSize
```

```
var_Items1 := var_Item:Items
```

```
var_Items1.PopupAppearance := ShadowBorder
```

```
var_Items1.Add("Item 1",nil,nil)
```

```
var_Items1.Add("Item 2",nil,nil)
```

```
var_Items1.Add("Item 3",nil,nil)
```

```
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Items,var_Items1
```

```
oRibbon = ole_1.Object
```

```
var_Items = oRibbon.Items
```

```
var_Item = var_Items.Add("Button",2)
```

```

var_Item.ShowAsButton = 19 /*exShowAsSelectButton |
exShowAsButtonAutoSize*/
var_Items1 = var_Item.Items
var_Items1.PopupAppearance = 6
var_Items1.Add("Item 1")
var_Items1.Add("Item 2")
var_Items1.Add("Item 3")
oRibbon.Refresh()

```

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Variant voltems

Get **ComItems** to voltems

Handle holtems

Get Create (RefClass(cComItems)) to holtems

Set pvComObject of holtems to voltems

Variant voltem

Get ComAdd of holtems "Button" 2 Nothing to voltem

Handle holtem

Get Create (RefClass(cComItem)) to holtem

Set pvComObject of holtem to voltem

Set ComShowAsButton of holtem to (OLEexShowAsSelectButton +
OLEexShowAsButtonAutoSize)

Variant voltems1

Get **ComItems** of holtem to voltems1

Handle holtems1

Get Create (RefClass(cComItems)) to holtems1

Set pvComObject of holtems1 to voltems1

Set ComPopupAppearance of holtems1 to OLEShadowBorder

Get ComAdd of holtems1 "Item 1" Nothing Nothing to Nothing

Get ComAdd of holtems1 "Item 2" Nothing Nothing to Nothing

Get ComAdd of holtems1 "Item 3" Nothing Nothing to Nothing

Send Destroy to holtems1

Send Destroy to holtem

```
Send Destroy to holtems  
Send ComRefresh  
End_Procedure
```

XBase++

```
#include "AppEvent.ch"  
#include "ActiveX.ch"  
  
PROCEDURE Main  
    LOCAL oForm  
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL  
    LOCAL oltem  
    LOCAL oltems, oltems1  
    LOCAL oRibbon  
  
    oForm := XbpDialog():new( AppDesktop() )  
    oForm:drawingArea:clipChildren := .T.  
    oForm:create( ,, {100,100}, {640,480},,, .F. )  
    oForm:close := {|| PostAppEvent( xbeP_Quit )}  
  
    oRibbon := XbpActiveXControl():new( oForm:drawingArea )  
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-  
CFBE431702E2}*/  
    oRibbon:create(,, {10,60},{610,370} )  
  
    oltems := oRibbon:Items()  
    oltem := oltems:Add("Button",2)  
    oltem:ShowAsButton :=  
19/*exShowAsSelectButton+exShowAsButtonAutoSize*/  
    oltems1 := oltem:Items()  
    oltems1:PopupAppearance := 6/*ShadowBorder*/  
    oltems1:Add("Item 1")  
    oltems1:Add("Item 2")  
    oltems1:Add("Item 3")  
    oRibbon:Refresh()
```

oForm:Show()

DO WHILE nEvent != xbeP_Quit

 nEvent := AppEvent(@mp1, @mp2, @oXbp)

 oXbp:handleEvent(nEvent, mp1, mp2)

ENDDO

RETURN

property Item.ShowAsDisabled as Boolean

Specifies whether the item is shown as disabled.

Type	Description
Boolean	A Boolean expression that specifies whether the current item is shown as disabled.

By default, the ShowAsDisabled property is False. Use the ShowAsDisabled property to shows the current item as disabled. The [Enabled](#) property specifies whether the item is enabled or disabled. The ShowAsDisabled property does not change the Enabled property, the item acts like an enabled item. For instance, you can not highlight a disabled item, instead you can highlight an item that looks as disabled. A disabled item ([Enabled](#) property is False) shows as grayed, and it is un-selectable, so the user can select or highlight it. An item ([Enabled](#) property is True, ShowAsDisabled property is True), shows as grayed, but it is selectable, so the user can select or highlight it.

How can I show the item as disabled but still be able to select or use it?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    .Add("Item").ShowAsDisabled = True
    .Add("").ToString = "Item[showdis=1]"
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    .Add("Item").ShowAsDisabled = True
    .Add("").ToString = "Item[showdis=1]"
  End With
  .Refresh
End With
```

VB.NET

```
With ExRibbon1
```

```
With .Items
```

```
.Add("Item").ShowAsDisabled = True
```

```
.Add("").ToString = "Item[showdis=1]"
```

```
End With
```

```
.Refresh()
```

```
End With
```

VB.NET for /COM

```
With AxRibbon1
```

```
With .Items
```

```
.Add("Item").ShowAsDisabled = True
```

```
.Add("").ToString = "Item[showdis=1]"
```

```
End With
```

```
.Refresh()
```

```
End With
```

C++

```
/*
```

```
Copy and paste the following directives to your header file as  
it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control  
Library'
```

```
#import <ExRibbon.dll>
```

```
using namespace EXRIBBONLib;
```

```
*/
```

```
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
```

```
>GetControlUnknown();
```

```
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
```

```
var_Items->Add(L"Item",vtMissing,vtMissing)-
```

```
> PutShowAsDisabled(VARIANT_TRUE);
```

```
var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"Item[showdis=1]");
```

```
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;  
    var_Items->Add(L"Item",TNoParam(),TNoParam())->ShowAsDisabled = true;  
    var_Items->Add(L"",TNoParam(),TNoParam())->ToString = L"Item[showdis=1]";  
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
    var_Items.Add("Item",null,null).ShowAsDisabled = true;  
    var_Items.Add("",null,null).ToString = "Item[showdis=1]";  
exribbon1.Refresh();
```

JSript/JavaScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    var var_Items = Ribbon1.Items;  
    var_Items.Add("Item",null,null).ShowAsDisabled = true;  
    var_Items.Add("",null,null).ToString = "Item[showdis=1]";  
    Ribbon1.Refresh();  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>
```



```

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Ribbon1
    With .Items
      .Add("Item").ShowAsDisabled = True
      .Add("").ToString = "Item[showdis=1]"
    End With
  .Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.Add("Item",null,null).ShowAsDisabled = true;
var_Items.Add("",null,null).ToString = "Item[showdis=1]";
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
  COM com_Item,com_Items;
  anytype var_Item,var_Items;
  ;

  super();

  var_Items = exribbon1.Items(); com_Items = var_Items;
  var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
  com_Item.ShowAsDisabled(true);
  var_Item = COM::createFromObject(com_Items.Add("")); com_Item = var_Item;
  com_Item.ToString("Item[showdis=1]");
}

```

```
exribbon1.Refresh();  
}
```

Delphi 8 (.NET only)

```
with AxRibbon1 do  
begin  
  with Items do  
  begin  
    Add('Item',Nil,Nil).ShowAsDisabled := True;  
    Add('',Nil,Nil).ToString := 'Item[showdis=1]';  
  end;  
  Refresh();  
end
```

Delphi (standard)

```
with Ribbon1 do  
begin  
  with Items do  
  begin  
    Add('Item',Null,Null).ShowAsDisabled := True;  
    Add('',Null,Null).ToString := 'Item[showdis=1]';  
  end;  
  Refresh();  
end
```

VFP

```
with thisform.Ribbon1  
  with .Items  
    .Add("Item").ShowAsDisabled = .T.  
    .Add("").ToString = "Item[showdis=1]"  
  endwith  
  .Refresh  
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Item1,var_Items
```

```
oRibbon = form.ActiveX1.nativeObject
```

```
var_Items = oRibbon.Items
```

```
// var_Items.Add("Item").ShowAsDisabled = true
```

```
var_Item = var_Items.Add("Item")
```

```
with (oRibbon)
```

```
    TemplateDef = [Dim var_Item]
```

```
    TemplateDef = var_Item
```

```
    Template = [var_Item.ShowAsDisabled = true]
```

```
endwith
```

```
// var_Items.Add("").ToString = "Item[showdis=1]"
```

```
var_Item1 = var_Items.Add("")
```

```
with (oRibbon)
```

```
    TemplateDef = [Dim var_Item1]
```

```
    TemplateDef = var_Item1
```

```
    Template = [var_Item1.ToString = "Item[showdis=1]"]
```

```
endwith
```

```
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
```

```
Dim var_Item as P
```

```
Dim var_Item1 as P
```

```
Dim var_Items as P
```

```
oRibbon = topparent:CONTROL_ACTIVEX1.activex
```

```
var_Items = oRibbon.Items
```

```
' var_Items.Add("Item").ShowAsDisabled = .t.
```

```
var_Item = var_Items.Add("Item")
```

```
oRibbon.TemplateDef = "Dim var_Item"
```

```
oRibbon.TemplateDef = var_Item
```

```
oRibbon.Template = "var_Item.ShowAsDisabled = True"
```

```
' var_Items.Add("").ToString = "Item[showdis=1]"
```

```
var_Item1 = var_Items.Add("")
oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1
oRibbon.Template = "var_Item1.ToString = \"Item[showdis=1]\"

oRibbon.Refresh()
```

Visual Objects

```
local var_Items as Items

var_Items := oDCOCX_Exontrol1:Items
var_Items.Add("Item",nil,nil):ShowAsDisabled := true
var_Items.Add("",nil,nil):ToString := "Item[showdis=1]"
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Items

oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Items.Add("Item").ShowAsDisabled = true
var_Items.Add("").ToString = "Item[showdis=1]"
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
    Variant voltems
    Get ComItems to voltems
    Handle holtems
    Get Create (RefClass(cComItems)) to holtems
    Set pvComObject of holtems to voltems
```

```

Variant voltem
Get ComAdd of holtems "Item" Nothing Nothing to voltem
Handle holtem
Get Create (RefClass(cComItem)) to holtem
Set pvComObject of holtem to voltem
    Set ComShowAsDisabled of holtem to True
Send Destroy to holtem
Variant voltem1
Get ComAdd of holtems "" Nothing Nothing to voltem1
Handle holtem1
Get Create (RefClass(cComItem)) to holtem1
Set pvComObject of holtem1 to voltem1
    Set ComToString of holtem1 to "Item[showdis= 1]"
Send Destroy to holtem1
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/

```

```
oRibbon:create(,, {10,60},{610,370} )
```

```
oItems := oRibbon:Items()
```

```
oItems:Add("Item"):ShowAsDisabled := .T.
```

```
oItems:Add(""):ToString := "Item[showdis=1]"
```

```
oRibbon:Refresh()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

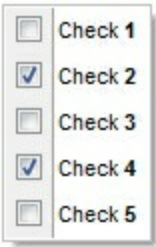
property Item.ShowCheckedAsSelected as ShowCheckedAsSelectedEnum

Specifies whether the checked item shows as selected.

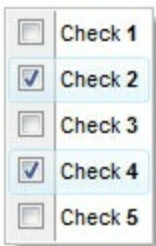
Type	Description
ShowCheckedAsSelectedEnum	A ShowCheckedAsSelectedEnum expression that specifies whether the checked item shows as selected.

By default, the ShowCheckedAsSelected property is exDisplayItemCheckInherit, which indicates that the control's [ShowCheckedAsSelected](#) property specifies how the checked item is shown. Use the ShowCheckedAsSelected property on non zero, to show the checked items as selected. A checked item is an item with the [Check](#) or [Radio](#) property set on True and the [Checked](#) property is True. The [SelBackColor](#) property indicates the color to show background of the selected / highlighted item. The [AllowToggleRadio](#) property on True, allows a radio button to set on zero (unchecked), if the user clicks twice the radio button.

The following screen shot shows the control when the ShowCheckedAsSelected property is exDisplayItemCheckDefault(by default):



The following screen shot shows the control when the ShowCheckedAsSelected property is exDisplayItemCheckHighlight:



How can I add check-buttons to items, without showing the check-box (method 2)?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add(,2)
```

```

.GroupPopup = 3 ' GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
With .Items
    With .Add("Check 1")
        .Check = True
        .Checked = True
        .ShowCheckedAsSelected = 1
    End With
With .Add("Check 2")
    .Check = True
    .ShowCheckedAsSelected = 1
End With
With .Add("Check 3")
    .Check = True
    .Checked = True
    .ShowCheckedAsSelected = 1
End With
End With
End With
End With
.Refresh
End With

```

VB6

```

With Ribbon1
    With .Items
        With .Add("",2)
            .GroupPopup = GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
        With .Items
            With .Add("Check 1")
                .Check = True
                .Checked = True
                .ShowCheckedAsSelected = exDisplayItemHighlight
            End With
            With .Add("Check 2")

```



```

        .Check = True
        .ShowCheckedAsSelected = exDisplayItemHighlight
    End With
    With .Add("Check 3")
        .Check = True
        .Checked = True
        .ShowCheckedAsSelected = exDisplayItemHighlight
    End With
End With
End With
End With
.Refresh
End With

```

VB.NET

```

With Exribbon1
    With .Items
        With .Add("",2)
            .GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup
        With .Items
            With .Add("Check 1")
                .Check = True
                .Checked = True
                .ShowCheckedAsSelected =
exontrol.EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight
            End With
            With .Add("Check 2")
                .Check = True
                .ShowCheckedAsSelected =
exontrol.EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight
            End With
            With .Add("Check 3")
                .Check = True
                .Checked = True
            End With
        End With
    End With
End With

```

.ShowCheckedAsSelected =

```
exontrol.EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight  
    End With  
    End With  
    End With  
    End With  
    .Refresh()  
End With
```

VB.NET for /COM

```
With AxRibbon1  
    With .Items  
        With .Add("",2)  
            .GroupPopup = EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or  
EXRIBBONLib.GroupPopupEnum.exGroupPopup  
            With .Items  
                With .Add("Check 1")  
                    .Check = True  
                    .Checked = True  
                    .ShowCheckedAsSelected =  
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight  
                End With  
                With .Add("Check 2")  
                    .Check = True  
                    .ShowCheckedAsSelected =  
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight  
                End With  
                With .Add("Check 3")  
                    .Check = True  
                    .Checked = True  
                    .ShowCheckedAsSelected =  
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight  
                End With  
            End With  
        End With  
    End With  
End With
```

.Refresh()
End With

C++

```
/*  
    Copy and paste the following directives to your header file as  
    it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control  
    Library'  
  
    #import <ExRibbon.dll>  
    using namespace EXRIBBONLib;  
*/  
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-  
> GetControlUnknown();  
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();  
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"",long(2),vtMissing);  
    var_Item->  
> PutGroupPopup(EXRIBBONLib::GroupPopupEnum(EXRIBBONLib::exNoGroupPopupF  
| EXRIBBONLib::exGroupPopup));  
    EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();  
    EXRIBBONLib::IItemPtr var_Item1 = var_Items1->Add(L"Check  
1",vtMissing,vtMissing);  
    var_Item1->PutCheck(VARIANT_TRUE);  
    var_Item1->PutChecked(VARIANT_TRUE);  
    var_Item1->  
> PutShowCheckedAsSelected(EXRIBBONLib::exDisplayItemHighlight);  
    EXRIBBONLib::IItemPtr var_Item2 = var_Items1->Add(L"Check  
2",vtMissing,vtMissing);  
    var_Item2->PutCheck(VARIANT_TRUE);  
    var_Item2->  
> PutShowCheckedAsSelected(EXRIBBONLib::exDisplayItemHighlight);  
    EXRIBBONLib::IItemPtr var_Item3 = var_Items1->Add(L"Check  
3",vtMissing,vtMissing);  
    var_Item3->PutCheck(VARIANT_TRUE);  
    var_Item3->PutChecked(VARIANT_TRUE);  
    var_Item3->
```

```
> PutShowCheckedAsSelected(EXRIBBONLib::exDisplayItemHighlight);  
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;  
    Exribbonlib_tlb::IItemPtr var_Item = var_Items->Add(L"",TVariant(2),TNoParam());  
    var_Item->GroupPopup =  
Exribbonlib_tlb::GroupPopupEnum::exNoGroupPopupFrame |  
Exribbonlib_tlb::GroupPopupEnum::exGroupPopup;  
    Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;  
    Exribbonlib_tlb::IItemPtr var_Item1 = var_Items1->Add(L"Check  
1",TNoParam(),TNoParam());  
    var_Item1->Check = true;  
    var_Item1->Checked = true;  
    var_Item1->ShowCheckedAsSelected =  
Exribbonlib_tlb::ShowCheckedAsSelectedEnum::exDisplayItemHighlight;  
    Exribbonlib_tlb::IItemPtr var_Item2 = var_Items1->Add(L"Check  
2",TNoParam(),TNoParam());  
    var_Item2->Check = true;  
    var_Item2->ShowCheckedAsSelected =  
Exribbonlib_tlb::ShowCheckedAsSelectedEnum::exDisplayItemHighlight;  
    Exribbonlib_tlb::IItemPtr var_Item3 = var_Items1->Add(L"Check  
3",TNoParam(),TNoParam());  
    var_Item3->Check = true;  
    var_Item3->Checked = true;  
    var_Item3->ShowCheckedAsSelected =  
Exribbonlib_tlb::ShowCheckedAsSelectedEnum::exDisplayItemHighlight;  
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
    exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);  
    var_Item.GroupPopup =  
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
```

```

exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup;
    exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
        exontrol.EXRIBBONLib.Item var_Item1 = var_Items1.Add("Check 1",null,null);
            var_Item1.Check = true;
            var_Item1.Checked = true;
            var_Item1.ShowCheckedAsSelected =
exontrol.EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
        exontrol.EXRIBBONLib.Item var_Item2 = var_Items1.Add("Check 2",null,null);
            var_Item2.Check = true;
            var_Item2.ShowCheckedAsSelected =
exontrol.EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
        exontrol.EXRIBBONLib.Item var_Item3 = var_Items1.Add("Check 3",null,null);
            var_Item3.Check = true;
            var_Item3.Checked = true;
            var_Item3.ShowCheckedAsSelected =
exontrol.EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload= 'Init()' >
<OBJECT CLASSID= "clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id= "Ribbon1" > </OBJECT>

<SCRIPT LANGUAGE= "JScript" >
function Init()
{
    var var_Items = Ribbon1.Items;
    var var_Item = var_Items.Add("",2,null);
    var_Item.GroupPopup = 3;
    var var_Items1 = var_Item.Items;
    var var_Item1 = var_Items1.Add("Check 1",null,null);
    var_Item1.Check = true;
    var_Item1.Checked = true;
    var_Item1.ShowCheckedAsSelected = 1;
    var var_Item2 = var_Items1.Add("Check 2",null,null);

```

```

        var_Item2.Check = true;
        var_Item2.ShowCheckedAsSelected = 1;
    var var_Item3 = var_Items1.Add("Check 3",null,null);
        var_Item3.Check = true;
        var_Item3.Checked = true;
        var_Item3.ShowCheckedAsSelected = 1;
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            With .Add("",2)
                .GroupPopup = 3 ' GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
            With .Items
                With .Add("Check 1")
                    .Check = True
                    .Checked = True
                    .ShowCheckedAsSelected = 1
                End With
                With .Add("Check 2")
                    .Check = True
                    .ShowCheckedAsSelected = 1
                End With
                With .Add("Check 3")
                    .Check = True

```

```

        .Checked = True
        .ShowCheckedAsSelected = 1
    End With
End With
End With
End With
.Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
    EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
    var_Item.GroupPopup =
EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
EXRIBBONLib.GroupPopupEnum.exGroupPopup;
    EXRIBBONLib.Items var_Items1 = var_Item.Items;
    EXRIBBONLib.Item var_Item1 = var_Items1.Add("Check 1",null,null);
    var_Item1.Check = true;
    var_Item1.Checked = true;
    var_Item1.ShowCheckedAsSelected =
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
    EXRIBBONLib.Item var_Item2 = var_Items1.Add("Check 2",null,null);
    var_Item2.Check = true;
    var_Item2.ShowCheckedAsSelected =
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
    EXRIBBONLib.Item var_Item3 = var_Items1.Add("Check 3",null,null);
    var_Item3.Check = true;
    var_Item3.Checked = true;
    var_Item3.ShowCheckedAsSelected =
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Item,com_Item1,com_Item2,com_Item3,com_Items,com_Items1;
    anytype var_Item,var_Item1,var_Item2,var_Item3,var_Items,var_Items1;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("",COMVariant::createFromInt(2)); com_Item =
var_Item;
    com_Item.GroupPopup(3/*exNoGroupPopupFrame | exGroupPopup*/);
    var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
    var_Item1 = com_Items1.Add("Check 1"); com_Item1 = var_Item1;
    com_Item1.Check(true);
    com_Item1.Checked(true);
    com_Item1.ShowCheckedAsSelected(1/*exDisplayItemHighlight*/);
    var_Item2 = com_Items1.Add("Check 2"); com_Item2 = var_Item2;
    com_Item2.Check(true);
    com_Item2.ShowCheckedAsSelected(1/*exDisplayItemHighlight*/);
    var_Item3 = com_Items1.Add("Check 3"); com_Item3 = var_Item3;
    com_Item3.Check(true);
    com_Item3.Checked(true);
    com_Item3.ShowCheckedAsSelected(1/*exDisplayItemHighlight*/);
    exribbon1.Refresh();
}
```

Delphi 8 (.NET only)

```
with AxRibbon1 do
begin
    with Items do
    begin
        with Add('',TObject(2),Nil) do
        begin
            GroupPopup :=
```



```

Integer(EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib.GroupPopupEnum.exGroupPopup);
    with Items do
    begin
        with Add('Check 1',Nil,Nil) do
        begin
            Check := True;
            Checked := True;
            ShowCheckedAsSelected :=
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
        end;
        with Add('Check 2',Nil,Nil) do
        begin
            Check := True;
            ShowCheckedAsSelected :=
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
        end;
        with Add('Check 3',Nil,Nil) do
        begin
            Check := True;
            Checked := True;
            ShowCheckedAsSelected :=
EXRIBBONLib.ShowCheckedAsSelectedEnum.exDisplayItemHighlight;
        end;
    end;
end;
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin
        with Add('',OleVariant(2),Null) do

```

```

begin
    GroupPopup := Integer(EXRIBBONLib_TLB.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib_TLB.exGroupPopup);
    with Items do
        begin
            with Add('Check 1',Null,Null) do
                begin
                    Check := True;
                    Checked := True;
                    ShowCheckedAsSelected := EXRIBBONLib_TLB.exDisplayItemHighlight;
                end;
            with Add('Check 2',Null,Null) do
                begin
                    Check := True;
                    ShowCheckedAsSelected := EXRIBBONLib_TLB.exDisplayItemHighlight;
                end;
            with Add('Check 3',Null,Null) do
                begin
                    Check := True;
                    Checked := True;
                    ShowCheckedAsSelected := EXRIBBONLib_TLB.exDisplayItemHighlight;
                end;
            end;
        end;
    end;
end;
Refresh();
end

```

VFP

```

with thisform.Ribbon1
    with .Items
        with Add("",2)
            .GroupPopup = 3 && GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
        with .Items
            with Add("Check 1")

```

```

        .Check = .T.
        .Checked = .T.
        .ShowCheckedAsSelected = 1
    endwhile
    with .Add("Check 2")
        .Check = .T.
        .ShowCheckedAsSelected = 1
    endwhile
    with .Add("Check 3")
        .Check = .T.
        .Checked = .T.
        .ShowCheckedAsSelected = 1
    endwhile
endwith
endwith
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Item2,var_Item3,var_Items,var_Items1

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
var_Items1 = var_Item.Items
var_Item1 = var_Items1.Add("Check 1")
var_Item1.Check = true
var_Item1.Checked = true
var_Item1.ShowCheckedAsSelected = 1
var_Item2 = var_Items1.Add("Check 2")
var_Item2.Check = true
var_Item2.ShowCheckedAsSelected = 1
var_Item3 = var_Items1.Add("Check 3")
var_Item3.Check = true

```

```
var_Item3.Checked = true
var_Item3.ShowCheckedAsSelected = 1
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Item2 as P
Dim var_Item3 as P
Dim var_Items as P
Dim var_Items1 as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 3 'exNoGroupPopupFrame + exGroupPopup
var_Items1 = var_Item.Items
var_Item1 = var_Items1.Add("Check 1")
var_Item1.Check = .t.
var_Item1.Checked = .t.
var_Item1.ShowCheckedAsSelected = 1
var_Item2 = var_Items1.Add("Check 2")
var_Item2.Check = .t.
var_Item2.ShowCheckedAsSelected = 1
var_Item3 = var_Items1.Add("Check 3")
var_Item3.Check = .t.
var_Item3.Checked = .t.
var_Item3.ShowCheckedAsSelected = 1
oRibbon.Refresh()
```

Visual Objects

```
local var_Item,var_Item1,var_Item2,var_Item3 as IItem
local var_Items,var_Items1 as IItems
```

```

var_Items := oDCOCX_Exontrol1:Items
var_Item := var_Items:Add("",2,nil)
var_Item:GroupPopup := exNoGroupPopupFrame | exGroupPopup
var_Items1 := var_Item:Items
var_Item1 := var_Items1:Add("Check 1",nil,nil)
var_Item1:Check := true
var_Item1:Checked := true
var_Item1:ShowCheckedAsSelected := exDisplayItemHighlight
var_Item2 := var_Items1:Add("Check 2",nil,nil)
var_Item2:Check := true
var_Item2:ShowCheckedAsSelected := exDisplayItemHighlight
var_Item3 := var_Items1:Add("Check 3",nil,nil)
var_Item3:Check := true
var_Item3:Checked := true
var_Item3:ShowCheckedAsSelected := exDisplayItemHighlight
oDCOCX_Exontrol1:Refresh()

```

PowerBuilder

```

OleObject oRibbon,var_Item,var_Item1,var_Item2,var_Item3,var_Items,var_Items1

oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
var_Items1 = var_Item.Items
var_Item1 = var_Items1.Add("Check 1")
var_Item1.Check = true
var_Item1.Checked = true
var_Item1.ShowCheckedAsSelected = 1
var_Item2 = var_Items1.Add("Check 2")
var_Item2.Check = true
var_Item2.ShowCheckedAsSelected = 1
var_Item3 = var_Items1.Add("Check 3")
var_Item3.Check = true

```

```
var_Item3.Checked = true
var_Item3.ShowCheckedAsSelected = 1
oRibbon.Refresh()
```

Visual DataFlex

Procedure OnCreate

```
Forward Send OnCreate
Variant voltems
Get ComItems to voltems
Handle holtems
Get Create (RefClass(cComItems)) to holtems
Set pvComObject of holtems to voltems
Variant voltem
Get ComAdd of holtems "" 2 Nothing to voltem
Handle holtem
Get Create (RefClass(cComItem)) to holtem
Set pvComObject of holtem to voltem
Set ComGroupPopup of holtem to (OLEexNoGroupPopupFrame +
OLEexGroupPopup)
Variant voltems1
Get ComItems of holtem to voltems1
Handle holtems1
Get Create (RefClass(cComItems)) to holtems1
Set pvComObject of holtems1 to voltems1
Variant voltem1
Get ComAdd of holtems1 "Check 1" Nothing Nothing to voltem1
Handle holtem1
Get Create (RefClass(cComItem)) to holtem1
Set pvComObject of holtem1 to voltem1
Set ComCheck of holtem1 to True
Set ComChecked of holtem1 to True
Set ComShowCheckedAsSelected of holtem1 to
OLEexDisplayItemHighlight
Send Destroy to holtem1
Variant voltem2
```

```

    Get ComAdd of holtems1 "Check 2" Nothing Nothing to voltem2
    Handle holtem2
    Get Create (RefClass(cComItem)) to holtem2
    Set pvComObject of holtem2 to voltem2
        Set ComCheck of holtem2 to True
        Set ComShowCheckedAsSelected of holtem2 to
OLEexDisplayItemHighlight
    Send Destroy to holtem2
    Variant voltem3
    Get ComAdd of holtems1 "Check 3" Nothing Nothing to voltem3
    Handle holtem3
    Get Create (RefClass(cComItem)) to holtem3
    Set pvComObject of holtem3 to voltem3
        Set ComCheck of holtem3 to True
        Set ComChecked of holtem3 to True
        Set ComShowCheckedAsSelected of holtem3 to
OLEexDisplayItemHighlight
    Send Destroy to holtem3
    Send Destroy to holtems1
    Send Destroy to holtem
    Send Destroy to holtems
    Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem,oltem1,oltem2,oltem3
    LOCAL oltems,oltems1
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )

```

```

oForm:drawingArea:clipChildren := .T.
oForm:create( ,, {100,100}, {640,480},, .F. )
oForm:close := {|| PostAppEvent( xbeP_Quit )}

oRibbon := XbpActiveXControl():new( oForm:drawingArea )
oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
oRibbon:create(,, {10,60},{610,370} )

oltems := oRibbon:Items()
oltem := oltems:Add("",2)
oltem:GroupPopup := 3/*exNoGroupPopupFrame+exGroupPopup*/
oltems1 := oltem:Items()
oltem1 := oltems1:Add("Check 1")
oltem1:Check := .T.
oltem1:Checked := .T.
oltem1:ShowCheckedAsSelected := 1/*exDisplayItemHighlight*/
oltem2 := oltems1:Add("Check 2")
oltem2:Check := .T.
oltem2:ShowCheckedAsSelected := 1/*exDisplayItemHighlight*/
oltem3 := oltems1:Add("Check 3")
oltem3:Check := .T.
oltem3:Checked := .T.
oltem3:ShowCheckedAsSelected := 1/*exDisplayItemHighlight*/
oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```


property Item.ShowLocalPopup as Boolean

Specifies whether the item's popup is shown as local. Clicking any item inside a local popup makes the popup itself to close including all its descendent sub-menus, without closing any ascendant sub-menus.

Type	Description
Boolean	A Boolean expression that specifies whether the item's sub-menu is closed when user clicks an item.

By default, the ShowLocalPopup property is False. Use the ShowLocalPopup property on True, to provide drop down list. Clicking any item inside a local popup makes the popup itself to close including all its descendent sub-menus, without closing any ascendant sub-menus. The [LocalAppearance](#) property specifies a different visual appearance for the local popup. Use the [CloseOnClick](#) property to specify how to close the current popup when user clicks a specified item. The [PopupAppearance](#) specifies a different visual appearance for the current submenu.

property Item.ShowPopupAlign as ShowPopupAlignEnum

Retrieves or sets a value that indicates how the item's sub-menu is aligned relative to the parent item.

Type	Description
ShowPopupAlignEnum	A ShowPopupAlignEnum expression that specifies whether the sub-menu is shown down or up to the item

By default, the ShowDown property is exShowPopupAlignDown + exShowPopupAlignRight. Use the ShowDown property to show the sub-menu up to the item. The [SubMenu/Items](#) property accesses the collection of Item objects to be shown on the sub-menu.

Is it possible to show the popup bellow to the item, rather than on the right side?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("Popup",2)
      .ShowPopupAlign = 1
      .ShowPopupArrow = False
    With .Items
      .PopupAppearance = 1
      .Add "Item 1"
      .Add "Item 2"
      .Add "Item 3"
    End With
  End With
End With
.Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    With .Add("Popup",2)
      .ShowPopupAlign = exShowPopupAlignDown
      .ShowPopupArrow = False
    End With
  End With
End With
```

```

    With .Items
        .PopupAppearance = FlatBorder
        .Add "Item 1"
        .Add "Item 2"
        .Add "Item 3"
    End With
End With
End With
.Refresh
End With

```

VB.NET

```

With Exribbon1
    With .Items
        With .Add("Popup",2)
            .ShowPopupAlign =
exontrol.EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown
            .ShowPopupArrow = False
        End With
        .PopupAppearance = exontrol.EXRIBBONLib.AppearanceEnum.FlatBorder
        .Add("Item 1")
        .Add("Item 2")
        .Add("Item 3")
    End With
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    With .Items
        With .Add("Popup",2)
            .ShowPopupAlign =
EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown
            .ShowPopupArrow = False
        End With
    End With
End With

```

```

    With .Items
        .PopupAppearance = EXRIBBONLib.AppearanceEnum.FlatBorder
        .Add("Item 1")
        .Add("Item 2")
        .Add("Item 3")
    End With
End With
End With
.Refresh()
End With

```

C++

```

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Popup",long(2),vtMissing);
    var_Item->PutShowPopupAlign(EXRIBBONLib::exShowPopupAlignDown);
    var_Item->PutShowPopupArrow(VARIANT_FALSE);
    EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();
    var_Items1->PutPopupAppearance(EXRIBBONLib::FlatBorder);
    var_Items1->Add(L"Item 1",vtMissing,vtMissing);
    var_Items1->Add(L"Item 2",vtMissing,vtMissing);
    var_Items1->Add(L"Item 3",vtMissing,vtMissing);
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;

```

```

Exribbonlib_tlb::IItemPtr var_Item = var_Items-
> Add(L"Popup",TVariant(2),TNoParam());
var_Item->ShowPopupAlign =
Exribbonlib_tlb::ShowPopupAlignEnum::exShowPopupAlignDown;
var_Item->ShowPopupArrow = false;
Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;
var_Items1->PopupAppearance =
Exribbonlib_tlb::AppearanceEnum::FlatBorder;
var_Items1->Add(L"Item 1",TNoParam(),TNoParam());
var_Items1->Add(L"Item 2",TNoParam(),TNoParam());
var_Items1->Add(L"Item 3",TNoParam(),TNoParam());
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Popup",2,null);
var_Item.ShowPopupAlign =
exontrol.EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown;
var_Item.ShowPopupArrow = false;
exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
var_Items1.PopupAppearance =
exontrol.EXRIBBONLib.AppearanceEnum.FlatBorder;
var_Items1.Add("Item 1",null,null);
var_Items1.Add("Item 2",null,null);
var_Items1.Add("Item 3",null,null);
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()

```

```

{
  var var_Items = Ribbon1.Items;
  var var_Item = var_Items.Add("Popup",2,null);
  var_Item.ShowPopupAlign = 1;
  var_Item.ShowPopupArrow = false;
  var var_Items1 = var_Item.Items;
  var_Items1.PopupAppearance = 1;
  var_Items1.Add("Item 1",null,null);
  var_Items1.Add("Item 2",null,null);
  var_Items1.Add("Item 3",null,null);
  Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Ribbon1
    With .Items
      With .Add("Popup",2)
        .ShowPopupAlign = 1
        .ShowPopupArrow = False
      With .Items
        .PopupAppearance = 1
        .Add "Item 1"
        .Add "Item 2"
        .Add "Item 3"
      End With
    End With
  End With
End With

```

```
.Refresh  
End With  
End Function  
</SCRIPT>  
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;  
    EXRIBBONLib.Item var_Item = var_Items.Add("Popup",2,null);  
    var_Item.ShowPopupAlign =  
EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown;  
    var_Item.ShowPopupArrow = false;  
    EXRIBBONLib.Items var_Items1 = var_Item.Items;  
        var_Items1.PopupAppearance = EXRIBBONLib.AppearanceEnum.FlatBorder;  
        var_Items1.Add("Item 1",null,null);  
        var_Items1.Add("Item 2",null,null);  
        var_Items1.Add("Item 3",null,null);  
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()  
{  
    COM com_Item,com_Items,com_Items1;  
    anytype var_Item,var_Items,var_Items1;  
    ;  
  
    super();  
  
    var_Items = exribbon1.Items(); com_Items = var_Items;  
    var_Item = com_Items.Add("Popup",COMVariant::createFromInt(2)); com_Item =  
var_Item;  
        com_Item.ShowPopupAlign(1/*exShowPopupAlignDown*/);  
        com_Item.ShowPopupArrow(false);  
        var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
```

```

        com_Items1.PopupAppearance(1/*FlatBorder*/);
        com_Items1.Add("Item 1");
        com_Items1.Add("Item 2");
        com_Items1.Add("Item 3");
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        with Add('Popup',TObject(2),Nil) do
        begin
            ShowPopupAlign :=
EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown;
            ShowPopupArrow := False;
            with Items do
            begin
                PopupAppearance := EXRIBBONLib.AppearanceEnum.FlatBorder;
                Add('Item 1',Nil,Nil);
                Add('Item 2',Nil,Nil);
                Add('Item 3',Nil,Nil);
            end;
        end;
    end;
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin
        with Add('Popup',OleVariant(2),Null) do
        begin

```



```

ShowPopupAlign := EXRIBBONLib_TLB.exShowPopupAlignDown;
ShowPopupArrow := False;
with Items do
begin
    PopupAppearance := EXRIBBONLib_TLB.FlatBorder;
    Add('Item 1',Null,Null);
    Add('Item 2',Null,Null);
    Add('Item 3',Null,Null);
end;
end;
end;
Refresh();
end

```

VFP

```

with thisform.Ribbon1
with .Items
with .Add("Popup",2)
    .ShowPopupAlign = 1
    .ShowPopupArrow = .F.
    with .Items
        .PopupAppearance = 1
        .Add("Item 1")
        .Add("Item 2")
        .Add("Item 3")
    endwith
endwith
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Items,var_Items1

oRibbon = form.ActiveX1.nativeObject
var_Items =oRibbon.Items

```

```

var_Item = var_Items.Add("Popup",2)
var_Item.ShowPopupAlign = 1
var_Item.ShowPopupArrow = false
var_Items1 = var_Item.Items
var_Items1.PopupAppearance = 1
var_Items1.Add("Item 1")
var_Items1.Add("Item 2")
var_Items1.Add("Item 3")
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Items as P
Dim var_Items1 as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Item = var_Items.Add("Popup",2)
var_Item.ShowPopupAlign = 1
var_Item.ShowPopupArrow = .f.
var_Items1 = var_Item.Items
var_Items1.PopupAppearance = 1
var_Items1.Add("Item 1")
var_Items1.Add("Item 2")
var_Items1.Add("Item 3")
oRibbon.Refresh()

```

Visual Objects

```

local var_Item as IItem
local var_Items,var_Items1 as IItems

var_Items := oDCOCX_Exontrol1.Items
var_Item := var_Items.Add("Popup",2,nil)

```

```
var_Item:ShowPopupAlign := exShowPopupAlignDown
var_Item:ShowPopupArrow := false
var_Items1 := var_Item:Items
    var_Items1:PopupAppearance := FlatBorder
    var_Items1:Add("Item 1",nil,nil)
    var_Items1:Add("Item 2",nil,nil)
    var_Items1:Add("Item 3",nil,nil)
oDCOCX_Exontrol1:Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Items,var_Items1

oRibbon = ole_1.Object
var_Items = oRibbon.Items
    var_Item = var_Items.Add("Popup",2)
        var_Item.ShowPopupAlign = 1
        var_Item.ShowPopupArrow = false
        var_Items1 = var_Item.Items
            var_Items1.PopupAppearance = 1
            var_Items1.Add("Item 1")
            var_Items1.Add("Item 2")
            var_Items1.Add("Item 3")
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
    Variant voltems
    Get ComItems to voltems
    Handle holtems
    Get Create (RefClass(cComItems)) to holtems
    Set pvComObject of holtems to voltems
        Variant voltem
        Get ComAdd of holtems "Popup" 2 Nothing to voltem
```

```

Handle holtem
Get Create (RefClass(cComItem)) to holtem
Set pvComObject of holtem to voltem
  Set ComShowPopupAlign of holtem to OLEExShowPopupAlignDown
  Set ComShowPopupArrow of holtem to False
Variant voltems1
Get ComItems of holtem to voltems1
Handle holtems1
Get Create (RefClass(cComItems)) to holtems1
Set pvComObject of holtems1 to voltems1
  Set ComPopupAppearance of holtems1 to OLEFlatBorder
  Get ComAdd of holtems1 "Item 1" Nothing Nothing to Nothing
  Get ComAdd of holtems1 "Item 2" Nothing Nothing to Nothing
  Get ComAdd of holtems1 "Item 3" Nothing Nothing to Nothing
Send Destroy to holtems1
Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oltem
  LOCAL oltems,oltems1
  LOCAL oRibbon

  oForm := XbpDialog():new( AppDesktop() )
  oForm:drawingArea:clipChildren := .T.
  oForm:create( „{100,100}, {640,480}„ .F. )
  oForm:close := {|| PostAppEvent( xbeP_Quit )}

```

```

oRibbon := XbpActiveXControl():new( oForm:drawingArea )
oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
oRibbon:create(, {10,60},{610,370} )

oltems := oRibbon:Items()
oltem := oltems:Add("Popup",2)
oltem:ShowPopupAlign := 1/*exShowPopupAlignDown*/
oltem:ShowPopupArrow := .F.
oltems1 := oltem:Items()
oltems1:PopupAppearance := 1/*FlatBorder*/
oltems1:Add("Item 1")
oltems1:Add("Item 2")
oltems1:Add("Item 3")
oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
  nEvent := AppEvent( @mp1, @mp2, @oXbp )
  oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

property Item.ShowPopupArrow as Boolean

Gets or sets a value that indicates whether an item that has a sub-menu shows or hides its popup arrow.

Type	Description
Boolean	A Boolean expression that specifies whether the item displays the popup arrow.

By default, the ShowPopupArrow property is True. Use the ShowPopupArrow property to hide the popup's arrow on the parent item.

Is it possible to hide the popup's arrow?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("Popup",2)
      .ShowPopupArrow = False
    With .Items
      .PopupAppearance = 6
      .Add "Item 1"
      .Add "Item 2"
      .Add "Item 3"
    End With
  End With
  .Add("").ToString = "Popup[arrow=0][popupapp=6](Item 1,Item 2,Item 3)"
End With
.Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    With .Add("Popup",2)
      .ShowPopupArrow = False
    With .Items
      .PopupAppearance = ShadowBorder
    End With
  End With
End With
```

```

        .Add "Item 1"
        .Add "Item 2"
        .Add "Item 3"
    End With
End With
.Add("").ToString = "Popup[arrow=0][popupapp=6](Item 1,Item 2,Item 3)"
End With
.Refresh
End With

```

VB.NET

```

With Exribbon1
    With .Items
        With .Add("Popup",2)
            .ShowPopupArrow = False
            With .Items
                .PopupAppearance =
exontrol.EXRIBBONLib.AppearanceEnum.ShadowBorder
                .Add("Item 1")
                .Add("Item 2")
                .Add("Item 3")
            End With
        End With
    End With
    .Add("").ToString = "Popup[arrow=0][popupapp=6](Item 1,Item 2,Item 3)"
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    With .Items
        With .Add("Popup",2)
            .ShowPopupArrow = False
            With .Items
                .PopupAppearance = EXRIBBONLib.AppearanceEnum.ShadowBorder
                .Add("Item 1")
            End With
        End With
    End With
End With

```

```

        .Add("Item 2")
        .Add("Item 3")
    End With
End With
.Add("").ToString = "Popup[arrow=0][popupapp=6](Item 1,Item 2,Item 3)"
End With
.Refresh()
End With

```

C++

```

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

#import <ExRibbon.dll>
using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Popup",long(2),vtMissing);
var_Item->PutShowPopupArrow(VARIANT_FALSE);
EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();
var_Items1->PutPopupAppearance(EXRIBBONLib::ShadowBorder);
var_Items1->Add(L"Item 1",vtMissing,vtMissing);
var_Items1->Add(L"Item 2",vtMissing,vtMissing);
var_Items1->Add(L"Item 3",vtMissing,vtMissing);
var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"Popup[arrow=0]
[popupapp=6](Item 1,Item 2,Item 3)");
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
Exribbonlib_tlb::IItemPtr var_Item = var_Items-

```



```

>Add(L"Popup",TVariant(2),TNoParam());
var_Item->ShowPopupArrow = false;
Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;
var_Items1->PopupAppearance =
Exribbonlib_tlb::AppearanceEnum::ShadowBorder;
var_Items1->Add(L"Item 1",TNoParam(),TNoParam());
var_Items1->Add(L"Item 2",TNoParam(),TNoParam());
var_Items1->Add(L"Item 3",TNoParam(),TNoParam());
var_Items->Add(L"",TNoParam(),TNoParam())->ToString = L"Popup[arrow=0]
[popupapp=6](Item 1,Item 2,Item 3)";
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Popup",2,null);
var_Item.ShowPopupArrow = false;
exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
var_Items1.PopupAppearance =
exontrol.EXRIBBONLib.AppearanceEnum.ShadowBorder;
var_Items1.Add("Item 1",null,null);
var_Items1.Add("Item 2",null,null);
var_Items1.Add("Item 3",null,null);
var_Items.Add("",null,null).ToString = "Popup[arrow=0][popupapp=6](Item 1,Item
2,Item 3)";
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{

```

```

var var_Items = Ribbon1.Items;
var var_Item = var_Items.Add("Popup",2,null);
var var_Item.ShowPopupArrow = false;
var var_Items1 = var_Item.Items;
var var_Items1.PopupAppearance = 6;
var var_Items1.Add("Item 1",null,null);
var var_Items1.Add("Item 2",null,null);
var var_Items1.Add("Item 3",null,null);
var var_Items.Add("",null,null).ToString = "Popup[arrow=0][popupapp=6](Item
1,Item 2,Item 3)";
Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Ribbon1
    With .Items
      With .Add("Popup",2)
        .ShowPopupArrow = False
      With .Items
        .PopupAppearance = 6
        .Add "Item 1"
        .Add "Item 2"
        .Add "Item 3"
      End With
    End With
  End With
  .Add("").ToString = "Popup[arrow=0][popupapp=6](Item 1,Item 2,Item 3)"
End With

```

```
.Refresh  
End With  
End Function  
</SCRIPT>  
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;  
EXRIBBONLib.Item var_Item = var_Items.Add("Popup",2,null);  
var_Item.ShowPopupArrow = false;  
EXRIBBONLib.Items var_Items1 = var_Item.Items;  
var_Items1.PopupAppearance =  
EXRIBBONLib.AppearanceEnum.ShadowBorder;  
var_Items1.Add("Item 1",null,null);  
var_Items1.Add("Item 2",null,null);  
var_Items1.Add("Item 3",null,null);  
var_Items.Add("",null,null).ToString = "Popup[arrow=0][popupapp=6](Item 1,Item  
2,Item 3)";  
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()  
{  
    COM com_Item,com_Item1,com_Items,com_Items1;  
    anytype var_Item,var_Item1,var_Items,var_Items1;  
    ;  
  
    super();  
  
    var_Items = exribbon1.Items(); com_Items = var_Items;  
    var_Item = com_Items.Add("Popup",COMVariant::createFromInt(2)); com_Item =  
var_Item;  
    com_Item.ShowPopupArrow(false);  
    var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
```

```

        com_Items1.PopupAppearance(6/*ShadowBorder*/);
        com_Items1.Add("Item 1");
        com_Items1.Add("Item 2");
        com_Items1.Add("Item 3");
        var_Item1 = COM::createFromObject(com_Items.Add("")); com_Item1 =
var_Item1;
        com_Item1.ToString("Popup[arrow=0][popupapp=6](Item 1,Item 2,Item 3)");
        exribbon1.Refresh();
    }

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        with Add('Popup',TObject(2),Nil) do
        begin
            ShowPopupArrow := False;
            with Items do
            begin
                PopupAppearance := EXRIBBONLib.AppearanceEnum.ShadowBorder;
                Add('Item 1',Nil,Nil);
                Add('Item 2',Nil,Nil);
                Add('Item 3',Nil,Nil);
            end;
        end;
        Add('',Nil,Nil).ToString := 'Popup[arrow=0][popupapp=6](Item 1,Item 2,Item 3)';
    end;
    Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin

```

```

with Add('Popup',OleVariant(2),Null) do
begin
  ShowPopupArrow := False;
  with Items do
  begin
    PopupAppearance := EXRIBBONLib_TLB.ShadowBorder;
    Add('Item 1',Null,Null);
    Add('Item 2',Null,Null);
    Add('Item 3',Null,Null);
  end;
end;
Add('',Null,Null).ToString := 'Popup[arrow=0][popupapp=6](Item 1,Item 2,Item
3)';
end;
Refresh();
end

```

VFP

```

with thisform.Ribbon1
with .Items
with .Add("Popup",2)
  .ShowPopupArrow = .F.
  with .Items
    .PopupAppearance = 6
    .Add("Item 1")
    .Add("Item 2")
    .Add("Item 3")
  endwith
endwith
.Add("").ToString = "Popup[arrow=0][popupapp=6](Item 1,Item 2,Item 3)"
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Items,var_Items1

```

```

oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("Popup",2)
var_Item.ShowPopupArrow = false
var_Items1 = var_Item.Items
var_Items1.PopupAppearance = 6
var_Items1.Add("Item 1")
var_Items1.Add("Item 2")
var_Items1.Add("Item 3")
// var_Items.Add("").ToString = "Popup[arrow=0][popupapp=6](Item 1,Item
2,Item 3)"
var_Item1 = var_Items.Add("")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.ToString = "Popup[arrow=0][popupapp=6](Item 1,Item
2,Item 3)"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P
Dim var_Items1 as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Item = var_Items.Add("Popup",2)
var_Item.ShowPopupArrow = .f.
var_Items1 = var_Item.Items
var_Items1.PopupAppearance = 6
var_Items1.Add("Item 1")

```

```

var_Items1.Add("Item 2")
var_Items1.Add("Item 3")
' var_Items.Add("").ToString = "Popup[arrow=0][popupapp=6](Item 1,Item
2,Item 3)"
var_Item1 = var_Items.Add("")
oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1
oRibbon.Template = "var_Item1.ToString = \"Popup[arrow=0][popupapp=6](Item
1,Item 2,Item 3)\\""

oRibbon.Refresh()

```

Visual Objects

```

local var_Item as Item
local var_Items,var_Items1 as Items

var_Items := oDCOCX_Exontrol1:Items
var_Item := var_Items.Add("Popup",2,nil)
var_Item:ShowPopupArrow := false
var_Items1 := var_Item:Items
var_Items1:PopupAppearance := ShadowBorder
var_Items1.Add("Item 1",nil,nil)
var_Items1.Add("Item 2",nil,nil)
var_Items1.Add("Item 3",nil,nil)
var_Items.Add("",nil,nil):ToString := "Popup[arrow=0][popupapp=6](Item 1,Item
2,Item 3)"
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```

OleObject oRibbon,var_Item,var_Items,var_Items1

oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Item = var_Items.Add("Popup",2)

```

```

var_Item.ShowPopupArrow = false
var_Items1 = var_Item.Items
    var_Items1.PopupAppearance = 6
    var_Items1.Add("Item 1")
    var_Items1.Add("Item 2")
    var_Items1.Add("Item 3")
var_Items.Add("").ToString = "Popup[arrow=0][popupapp=6](Item 1,Item 2,Item
3)"
oRibbon.Refresh()

```

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Variant voltems

Get ComItems to voltems

Handle holtems

Get Create (RefClass(cComItems)) to holtems

Set pvComObject of holtems to voltems

Variant voltem

Get ComAdd of holtems "Popup" 2 Nothing to voltem

Handle holtem

Get Create (RefClass(cComItem)) to holtem

Set pvComObject of holtem to voltem

Set **ComShowPopupArrow** of holtem to False

Variant voltems1

Get ComItems of holtem to voltems1

Handle holtems1

Get Create (RefClass(cComItems)) to holtems1

Set pvComObject of holtems1 to voltems1

Set ComPopupAppearance of holtems1 to OLEShadowBorder

Get ComAdd of holtems1 "Item 1" Nothing Nothing to Nothing

Get ComAdd of holtems1 "Item 2" Nothing Nothing to Nothing

Get ComAdd of holtems1 "Item 3" Nothing Nothing to Nothing

Send Destroy to holtems1

Send Destroy to holtem


```

Variant voltem1
Get ComAdd of holtems "" Nothing Nothing to voltem1
Handle holtem1
Get Create (RefClass(cComItem)) to holtem1
Set pvComObject of holtem1 to voltem1
    Set ComToString of holtem1 to "Popup[arrow=0][popupapp=6](Item 1,Item
2,Item 3)"
    Send Destroy to holtem1
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem
    LOCAL oltems, oltems1
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oltems := oRibbon:Items()
    oltem := oltems:Add("Popup",2)
    oltem:ShowPopupArrow := .F.

```

```
    oltems1 := oltem:Items()
    oltems1:PopupAppearance := 6/*ShadowBorder*/
    oltems1:Add("Item 1")
    oltems1:Add("Item 2")
    oltems1:Add("Item 3")
    oltems:Add(""):ToString := "Popup[arrow=0][popupapp=6](Item 1,Item 2,Item
3)"
    oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN
```

property Item.ShowPopupAt as Long

Specifies the identifier of the item where the current item's submenu/popup is displayed.

Type	Description
Long	A long expression that specifies the identifier of the item where the current popup should be displayed.

By default, the ShowPopupAt property is 0. If no item with specified identifier is found, the default item is used to display its submenu. Use the [ShowPopupOffset](#) property specifies the relative offset to show the item's sub menu.

How can I display the drop down popup to a different position?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("",2)
      .GroupPopup = 1
    With .Items
      With .Add("Popup 1",2,100)
        .ShowPopupAlign = 1
        .ShowPopupArrow = False
      With .Items
        .PopupAppearance = 3
        .Add "Item 1"
        .Add "Item 2"
        .Add "Item 3"
      End With
    End With
  End With
  With .Add("Popup 2",2,200)
    .ShowPopupAt = 100
    .ShowPopupArrow = False
  With .Items
    .PopupAppearance = 3
    .Add "Item 4"
    .Add "Item 5"
    .Add "Item 6"
  End With
End With
```

```
End With
End With
End With
End With
.Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    With .Add("",2)
      .GroupPopup = exGroupPopup
    With .Items
      With .Add("Popup 1",2,100)
        .ShowPopupAlign = exShowPopupAlignDown
        .ShowPopupArrow = False
      With .Items
        .PopupAppearance = RaisedBorder
        .Add "Item 1"
        .Add "Item 2"
        .Add "Item 3"
      End With
    End With
  End With
  With .Add("Popup 2",2,200)
    .ShowPopupAt = 100
    .ShowPopupArrow = False
    With .Items
      .PopupAppearance = RaisedBorder
      .Add "Item 4"
      .Add "Item 5"
      .Add "Item 6"
    End With
  End With
End With
End With
End With
End With
```

.Refresh
End With

VB.NET

```
With Exribbon1
  With .Items
    With .Add("",2)
      .GroupPopup = exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup
    With .Items
      With .Add("Popup 1",2,100)
        .ShowPopupAlign =
exontrol.EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown
        .ShowPopupArrow = False
      With .Items
        .PopupAppearance =
exontrol.EXRIBBONLib.AppearanceEnum.RaisedBorder
        .Add("Item 1")
        .Add("Item 2")
        .Add("Item 3")
      End With
    End With
  With .Add("Popup 2",2,200)
    .ShowPopupAt = 100
    .ShowPopupArrow = False
    With .Items
      .PopupAppearance =
exontrol.EXRIBBONLib.AppearanceEnum.RaisedBorder
      .Add("Item 4")
      .Add("Item 5")
      .Add("Item 6")
    End With
  End With
End With
End With
End With
End With
.Refresh()
```

VB.NET for /COM

```
With AxRibbon1
  With .Items
    With .Add("",2)
      .GroupPopup = EXRIBBONLib.GroupPopupEnum.exGroupPopup
    With .Items
      With .Add("Popup 1",2,100)
        .ShowPopupAlign =
EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown
        .ShowPopupArrow = False
      With .Items
        .PopupAppearance = EXRIBBONLib.AppearanceEnum.RaisedBorder
        .Add("Item 1")
        .Add("Item 2")
        .Add("Item 3")
      End With
    End With
  End With
  With .Add("Popup 2",2,200)
    .ShowPopupAt = 100
    .ShowPopupArrow = False
    With .Items
      .PopupAppearance = EXRIBBONLib.AppearanceEnum.RaisedBorder
      .Add("Item 4")
      .Add("Item 5")
      .Add("Item 6")
    End With
  End With
End With
End With
End With
End With
End With
.Refresh()
End With
```

```
/*
```

Copy and paste the following directives to your header file as it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control Library'

```
#import <ExRibbon.dll>
using namespace EXRIBBONLib;
```

```
*/
```

```
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"",long(2),vtMissing);
var_Item->PutGroupPopup(EXRIBBONLib::exGroupPopup);
EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();
EXRIBBONLib::IItemPtr var_Item1 = var_Items1->Add(L"Popup
1",long(2),long(100));
var_Item1->PutShowPopupAlign(EXRIBBONLib::exShowPopupAlignDown);
var_Item1->PutShowPopupArrow(VARIANT_FALSE);
EXRIBBONLib::IItemsPtr var_Items2 = var_Item1->GetItems();
var_Items2->PutPopupAppearance(EXRIBBONLib::RaisedBorder);
var_Items2->Add(L"Item 1",vtMissing,vtMissing);
var_Items2->Add(L"Item 2",vtMissing,vtMissing);
var_Items2->Add(L"Item 3",vtMissing,vtMissing);
EXRIBBONLib::IItemPtr var_Item2 = var_Items1->Add(L"Popup
2",long(2),long(200));
var_Item2->PutShowPopupAt(100);
var_Item2->PutShowPopupArrow(VARIANT_FALSE);
EXRIBBONLib::IItemsPtr var_Items3 = var_Item2->GetItems();
var_Items3->PutPopupAppearance(EXRIBBONLib::RaisedBorder);
var_Items3->Add(L"Item 4",vtMissing,vtMissing);
var_Items3->Add(L"Item 5",vtMissing,vtMissing);
var_Items3->Add(L"Item 6",vtMissing,vtMissing);
spRibbon1->Refresh();
```

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
Exribbonlib_tlb::IItemPtr var_Item = var_Items->Add(L"",TVariant(2),TNoParam());
var_Item->GroupPopup = Exribbonlib_tlb::GroupPopupEnum::exGroupPopup;
Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;
Exribbonlib_tlb::IItemPtr var_Item1 = var_Items1->Add(L"Popup
1",TVariant(2),TVariant(100));
var_Item1->ShowPopupAlign =
Exribbonlib_tlb::ShowPopupAlignEnum::exShowPopupAlignDown;
var_Item1->ShowPopupArrow = false;
Exribbonlib_tlb::IItemsPtr var_Items2 = var_Item1->Items;
var_Items2->PopupAppearance =
Exribbonlib_tlb::AppearanceEnum::RaisedBorder;
var_Items2->Add(L"Item 1",TNoParam(),TNoParam());
var_Items2->Add(L"Item 2",TNoParam(),TNoParam());
var_Items2->Add(L"Item 3",TNoParam(),TNoParam());
Exribbonlib_tlb::IItemPtr var_Item2 = var_Items1->Add(L"Popup
2",TVariant(2),TVariant(200));
var_Item2->ShowPopupAt = 100;
var_Item2->ShowPopupArrow = false;
Exribbonlib_tlb::IItemsPtr var_Items3 = var_Item2->Items;
var_Items3->PopupAppearance =
Exribbonlib_tlb::AppearanceEnum::RaisedBorder;
var_Items3->Add(L"Item 4",TNoParam(),TNoParam());
var_Items3->Add(L"Item 5",TNoParam(),TNoParam());
var_Items3->Add(L"Item 6",TNoParam(),TNoParam());
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
var_Item.GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup;
exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
exontrol.EXRIBBONLib.Item var_Item1 = var_Items1.Add("Popup 1",2,100);
var_Item1.ShowPopupAlign =

```



```

exontrol.EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown;
    var_Item1.ShowPopupArrow = false;
    exontrol.EXRIBBONLib.Items var_Items2 = var_Item1.Items;
    var_Items2.PopupAppearance =
exontrol.EXRIBBONLib.AppearanceEnum.RaisedBorder;
    var_Items2.Add("Item 1",null,null);
    var_Items2.Add("Item 2",null,null);
    var_Items2.Add("Item 3",null,null);
    exontrol.EXRIBBONLib.Item var_Item2 = var_Items1.Add("Popup 2",2,200);
    var_Item2.ShowPopupAt = 100;
    var_Item2.ShowPopupArrow = false;
    exontrol.EXRIBBONLib.Items var_Items3 = var_Item2.Items;
    var_Items3.PopupAppearance =
exontrol.EXRIBBONLib.AppearanceEnum.RaisedBorder;
    var_Items3.Add("Item 4",null,null);
    var_Items3.Add("Item 5",null,null);
    var_Items3.Add("Item 6",null,null);
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var var_Item = var_Items.Add("",2,null);
    var_Item.GroupPopup = 1;
    var var_Items1 = var_Item.Items;
    var var_Item1 = var_Items1.Add("Popup 1",2,100);
    var_Item1.ShowPopupAlign = 1;
    var_Item1.ShowPopupArrow = false;
    var var_Items2 = var_Item1.Items;

```

```

        var_Items2.PopupAppearance = 3;
        var_Items2.Add("Item 1",null,null);
        var_Items2.Add("Item 2",null,null);
        var_Items2.Add("Item 3",null,null);
    var var_Item2 = var_Items1.Add("Popup 2",2,200);
    var_Item2.ShowPopupAt = 100;
    var_Item2.ShowPopupArrow = false;
    var var_Items3 = var_Item2.Items;
        var_Items3.PopupAppearance = 3;
        var_Items3.Add("Item 4",null,null);
        var_Items3.Add("Item 5",null,null);
        var_Items3.Add("Item 6",null,null);
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            With .Add("",2)
                .GroupPopup = 1
            With .Items
                With .Add("Popup 1",2,100)
                    .ShowPopupAlign = 1
                    .ShowPopupArrow = False
                With .Items
                    .PopupAppearance = 3
                    .Add "Item 1"
                End With
            End With
        End With
    End With
End Function

```

```

        .Add "Item 2"
        .Add "Item 3"
    End With
End With
With .Add("Popup 2",2,200)
    .ShowPopupAt = 100
    .ShowPopupArrow = False
    With .Items
        .PopupAppearance = 3
        .Add "Item 4"
        .Add "Item 5"
        .Add "Item 6"
    End With
End With
End With
End With
End With
    .Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
var_Item.GroupPopup = EXRIBBONLib.GroupPopupEnum.exGroupPopup;
EXRIBBONLib.Items var_Items1 = var_Item.Items;
EXRIBBONLib.Item var_Item1 = var_Items1.Add("Popup 1",2,100);
var_Item1.ShowPopupAlign =
EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown;
var_Item1.ShowPopupArrow = false;
EXRIBBONLib.Items var_Items2 = var_Item1.Items;
var_Items2.PopupAppearance =
EXRIBBONLib.AppearanceEnum.RaisedBorder;

```

```

        var_Items2.Add("Item 1",null,null);
        var_Items2.Add("Item 2",null,null);
        var_Items2.Add("Item 3",null,null);
    EXRIBBONLib.Item var_Item2 = var_Items1.Add("Popup 2",2,200);
    var_Item2.ShowPopupAt = 100;
    var_Item2.ShowPopupArrow = false;
    EXRIBBONLib.Items var_Items3 = var_Item2.Items;
    var_Items3.PopupAppearance =
EXRIBBONLib.AppearanceEnum.RaisedBorder;
    var_Items3.Add("Item 4",null,null);
    var_Items3.Add("Item 5",null,null);
    var_Items3.Add("Item 6",null,null);
    axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM
    com_Item,com_Item1,com_Item2,com_Items,com_Items1,com_Items2,com_Items3;
    anytype var_Item,var_Item1,var_Item2,var_Items,var_Items1,var_Items2,var_Items3;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("",COMVariant::createFromInt(2)); com_Item =
var_Item;
    com_Item.GroupPopup(1/*exGroupPopup*/);
    var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
    var_Item1 = com_Items1.Add("Popup
1",COMVariant::createFromInt(2),COMVariant::createFromInt(100)); com_Item1 =
var_Item1;
    com_Item1.ShowPopupAlign(1/*exShowPopupAlignDown*/);
    com_Item1.ShowPopupArrow(false);
    var_Items2 = com_Item1.Items(); com_Items2 = var_Items2;

```

```

        com_Items2.PopupAppearance(3/*RaisedBorder*/);
        com_Items2.Add("Item 1");
        com_Items2.Add("Item 2");
        com_Items2.Add("Item 3");
        var_Item2 = com_Items1.Add("Popup
2",COMVariant::createFromInt(2),COMVariant::createFromInt(200)); com_Item2 =
var_Item2;
        com_Item2.ShowPopupAt(100);
        com_Item2.ShowPopupArrow(false);
        var_Items3 = com_Item2.Items(); com_Items3 = var_Items3;
        com_Items3.PopupAppearance(3/*RaisedBorder*/);
        com_Items3.Add("Item 4");
        com_Items3.Add("Item 5");
        com_Items3.Add("Item 6");
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        with Add('',TObject(2),Nil) do
        begin
            GroupPopup := EXRIBBONLib.GroupPopupEnum.exGroupPopup;
            with Items do
            begin
                with Add('Popup 1',TObject(2),TObject(100)) do
                begin
                    ShowPopupAlign :=
EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown;
                    ShowPopupArrow := False;
                    with Items do
                    begin
                        PopupAppearance := EXRIBBONLib.AppearanceEnum.RaisedBorder;
                        Add('Item 1',Nil,Nil);

```

```

        Add('Item 2',Nil,Nil);
        Add('Item 3',Nil,Nil);
    end;
end;
with Add('Popup 2',TObject(2),TObject(200)) do
begin
    ShowPopupAt := 100;
    ShowPopupArrow := False;
    with Items do
    begin
        PopupAppearance := EXRIBBONLib.AppearanceEnum.RaisedBorder;
        Add('Item 4',Nil,Nil);
        Add('Item 5',Nil,Nil);
        Add('Item 6',Nil,Nil);
    end;
    end;
end;
end;
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin
        with Add('',OleVariant(2),Null) do
        begin
            GroupPopup := EXRIBBONLib_TLB.exGroupPopup;
            with Items do
            begin
                with Add('Popup 1',OleVariant(2),OleVariant(100)) do
                begin
                    ShowPopupAlign := EXRIBBONLib_TLB.exShowPopupAlignDown;
                    ShowPopupArrow := False;

```

```

with Items do
begin
    PopupAppearance := EXRIBBONLib_TLB.RaisedBorder;
    Add('Item 1',Null,Null);
    Add('Item 2',Null,Null);
    Add('Item 3',Null,Null);
end;
end;
with Add('Popup 2',OleVariant(2),OleVariant(200)) do
begin
    ShowPopupAt := 100;
    ShowPopupArrow := False;
    with Items do
    begin
        PopupAppearance := EXRIBBONLib_TLB.RaisedBorder;
        Add('Item 4',Null,Null);
        Add('Item 5',Null,Null);
        Add('Item 6',Null,Null);
    end;
end;
end;
end;
end;
Refresh();
end

```

VFP

```

with thisform.Ribbon1
with .Items
with .Add("",2)
.GroupPopup = 1
with .Items
with .Add("Popup 1",2,100)
.ShowPopupAlign = 1
.ShowPopupArrow = .F.
with .Items

```

```

        .PopupAppearance = 3
        .Add("Item 1")
        .Add("Item 2")
        .Add("Item 3")
    endwhile
endwith
with .Add("Popup 2",2,200)
    .ShowPopupAt = 100
    .ShowPopupArrow = .F.
    with .Items
        .PopupAppearance = 3
        .Add("Item 4")
        .Add("Item 5")
        .Add("Item 6")
    endwhile
endwith
endwith
endwith
endwith
.Refresh
endwith

```

dBASE Plus

```

local
oRibbon,var_Item,var_Item1,var_Item2,var_Items,var_Items1,var_Items2,var_Items3

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 1
var_Items1 = var_Item.Items
var_Item1 = var_Items1.Add("Popup 1",2,100)
var_Item1.ShowPopupAlign = 1
var_Item1.ShowPopupArrow = false
var_Items2 = var_Item1.Items
var_Items2.PopupAppearance = 3

```



```

        var_Items2.Add("Item 1")
        var_Items2.Add("Item 2")
        var_Items2.Add("Item 3")
var_Item2 = var_Items1.Add("Popup 2",2,200)
        var_Item2.ShowPopupAt = 100
        var_Item2.ShowPopupArrow = false
        var_Items3 = var_Item2.Items
            var_Items3.PopupAppearance = 3
            var_Items3.Add("Item 4")
            var_Items3.Add("Item 5")
            var_Items3.Add("Item 6")
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Item2 as P
Dim var_Items as P
Dim var_Items1 as P
Dim var_Items2 as P
Dim var_Items3 as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
    var_Item = var_Items.Add("",2)
        var_Item.GroupPopup = 1
        var_Items1 = var_Item.Items
            var_Item1 = var_Items1.Add("Popup 1",2,100)
                var_Item1.ShowPopupAlign = 1
                var_Item1.ShowPopupArrow = .f.
                var_Items2 = var_Item1.Items
                    var_Items2.PopupAppearance = 3
                    var_Items2.Add("Item 1")
                    var_Items2.Add("Item 2")

```

```

        var_Item2.Add("Item 3")
var_Item2 = var_Items1.Add("Popup 2",2,200)
var_Item2.ShowPopupAt = 100
var_Item2.ShowPopupArrow = .f
var_Items3 = var_Item2.Items
    var_Items3.PopupAppearance = 3
    var_Items3.Add("Item 4")
    var_Items3.Add("Item 5")
    var_Items3.Add("Item 6")
oRibbon.Refresh()

```

Visual Objects

```

local var_Item,var_Item1,var_Item2 as Item
local var_Items,var_Items1,var_Items2,var_Items3 as Items

var_Items := oDCOCX_Exontrol1.Items
var_Item := var_Items.Add("",2,nil)
var_Item:GroupPopup := exGroupPopup
var_Items1 := var_Item.Items
var_Item1 := var_Items1.Add("Popup 1",2,100)
var_Item1.ShowPopupAlign := exShowPopupAlignDown
var_Item1.ShowPopupArrow := false
var_Items2 := var_Item1.Items
    var_Items2.PopupAppearance := RaisedBorder
    var_Items2.Add("Item 1",nil,nil)
    var_Items2.Add("Item 2",nil,nil)
    var_Items2.Add("Item 3",nil,nil)
var_Item2 := var_Items1.Add("Popup 2",2,200)
var_Item2.ShowPopupAt := 100
var_Item2.ShowPopupArrow := false
var_Items3 := var_Item2.Items
    var_Items3.PopupAppearance := RaisedBorder
    var_Items3.Add("Item 4",nil,nil)
    var_Items3.Add("Item 5",nil,nil)
    var_Items3.Add("Item 6",nil,nil)

```

```
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

OleObject

oRibbon,var_Item,var_Item1,var_Item2,var_Items,var_Items1,var_Items2,var_Items3

oRibbon = ole_1.Object

var_Items = oRibbon.Items

var_Item = var_Items.Add("",2)

var_Item.GroupPopup = 1

var_Items1 = var_Item.Items

var_Item1 = var_Items1.Add("Popup 1",2,100)

var_Item1.ShowPopupAlign = 1

var_Item1.ShowPopupArrow = false

var_Items2 = var_Item1.Items

var_Items2.PopupAppearance = 3

var_Items2.Add("Item 1")

var_Items2.Add("Item 2")

var_Items2.Add("Item 3")

var_Item2 = var_Items1.Add("Popup 2",2,200)

var_Item2.**ShowPopupAt** = 100

var_Item2.ShowPopupArrow = false

var_Items3 = var_Item2.Items

var_Items3.PopupAppearance = 3

var_Items3.Add("Item 4")

var_Items3.Add("Item 5")

var_Items3.Add("Item 6")

oRibbon.Refresh()

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Variant voltems

Get Comltems to voltems

Handle holtems

Get Create (RefClass(cComltems)) to holtems

Set pvComObject of holtems to voltems

Variant voltem

Get ComAdd of holtems "" 2 Nothing to voltem

Handle holtem

Get Create (RefClass(cComltem)) to holtem

Set pvComObject of holtem to voltem

Set ComGroupPopup of holtem to OLEexGroupPopup

Variant voltems1

Get Comltems of holtem to voltems1

Handle holtems1

Get Create (RefClass(cComltems)) to holtems1

Set pvComObject of holtems1 to voltems1

Variant voltem1

Get ComAdd of holtems1 "Popup 1" 2 100 to voltem1

Handle holtem1

Get Create (RefClass(cComltem)) to holtem1

Set pvComObject of holtem1 to voltem1

Set ComShowPopupAlign of holtem1 to OLEexShowPopupAlignDown

Set ComShowPopupArrow of holtem1 to False

Variant voltems2

Get Comltems of holtem1 to voltems2

Handle holtems2

Get Create (RefClass(cComltems)) to holtems2

Set pvComObject of holtems2 to voltems2

Set ComPopupAppearance of holtems2 to OLERaisedBorder

Get ComAdd of holtems2 "Item 1" Nothing Nothing to Nothing

Get ComAdd of holtems2 "Item 2" Nothing Nothing to Nothing

Get ComAdd of holtems2 "Item 3" Nothing Nothing to Nothing

Send Destroy to holtems2

Send Destroy to holtem1

Variant voltem2

Get ComAdd of holtems1 "Popup 2" 2 200 to voltem2

Handle holtem2

Get Create (RefClass(cComltem)) to holtem2

Set pvComObject of holtem2 to voltem2

```

Set ComShowPopupAt of holtem2 to 100
Set ComShowPopupArrow of holtem2 to False
Variant voltems3
Get ComItems of holtem2 to voltems3
Handle holtems3
Get Create (RefClass(cComItems)) to holtems3
Set pvComObject of holtems3 to voltems3
    Set ComPopupAppearance of holtems3 to OLERaisedBorder
    Get ComAdd of holtems3 "Item 4" Nothing Nothing to Nothing
    Get ComAdd of holtems3 "Item 5" Nothing Nothing to Nothing
    Get ComAdd of holtems3 "Item 6" Nothing Nothing to Nothing
Send Destroy to holtems3
Send Destroy to holtem2
Send Destroy to holtems1
Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem,oltem1,oltem2
    LOCAL oltems,oltems1,oltems2,oltems3
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( „{100,100}, {640,480}„, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )

```

```
oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-  
CFBE431702E2}*/
```

```
oRibbon:create(, {10,60},{610,370} )
```

```
oltems := oRibbon:Items()
```

```
oltem := oltems:Add("",2)
```

```
oltem:GroupPopup := 1/*exGroupPopup*/
```

```
oltems1 := oltem:Items()
```

```
oltem1 := oltems1:Add("Popup 1",2,100)
```

```
oltem1:ShowPopupAlign := 1/*exShowPopupAlignDown*/
```

```
oltem1:ShowPopupArrow := .F.
```

```
oltems2 := oltem1:Items()
```

```
oltems2:PopupAppearance := 3/*RaisedBorder*/
```

```
oltems2:Add("Item 1")
```

```
oltems2:Add("Item 2")
```

```
oltems2:Add("Item 3")
```

```
oltem2 := oltems1:Add("Popup 2",2,200)
```

```
oltem2:ShowPopupAt := 100
```

```
oltem2:ShowPopupArrow := .F.
```

```
oltems3 := oltem2:Items()
```

```
oltems3:PopupAppearance := 3/*RaisedBorder*/
```

```
oltems3:Add("Item 4")
```

```
oltems3:Add("Item 5")
```

```
oltems3:Add("Item 6")
```

```
oRibbon:Refresh()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Item.ShowPopupOffset as String

Specifies the offset (horizontal,vertical) to display the item's submenu/popup relative to its default position.

Type	Description
String	<p>A String expression that specifies the offset as a pair (horizontal, vertical) to display the item's submenu/popup relative to its default position.</p> <p>The horizontal/vertical can be one of the following:</p> <ul style="list-style-type: none">• number, indicating the x,y-offset relative to default position of the drop down panel. The number can be followed by a D character which indicates that the coordinate is giving in dots, rather than pixels, and depends on the current DPI settings. For instance, ShowPopupOffset = "12D,12D", indicates that the inner popup is moved by 18,18 pixels if the DPI settings is 150%, while if DPI settings is 100% the popup is moved by 12,12 pixels, relative to its current position. The numberD converted in pixels is $\text{number} * \text{DPI} / 100$.• number:control, indicating the x,y-offset relative to control.• number:screen, indicating the x,y-absolute position on the screen.

By default, the ShowPopupOffset property is empty (equivalent with the "0,0"). The ShowPopupOffset property specifies the offset as a pair (horizontal,vertical) to display the item's submenu/popup relative to its default position/control or screen. Use the [ShowPopupAt](#) property to specify the identifier of the item where the current item's submenu/popup is displayed.

For instance, the following sample, displays the item's drop down relative to the control (call ShowPopupOffset):

```
With Ribbon1
  .PopupAppearance = ShadowBorder
With .Items
  With .Add("Popup <b>1",2,100)
    .ShowPopupArrow = False
```

```
.ShowPopupAlign = exShowPopupAlignDown
```

```
.ShowPopupOffset = "0,0:control"
```

```
With .Items
```

```
  .Add "Item 1"
```

```
  .Add "Item 2"
```

```
  .Add "Item 3"
```

```
End With
```

```
End With
```

```
With .Add("Popup <b>2",2,200)
```

```
  .GroupPopup = exGroupPopup
```

```
With .Items
```

```
  .Add "Item 2"
```

```
  .Add "Item 3"
```

```
  .Add "Item 4"
```

```
End With
```

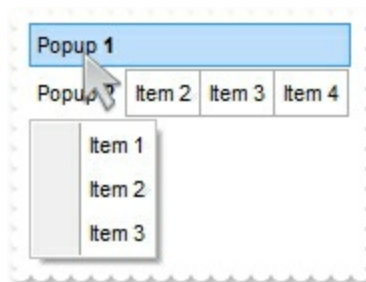
```
End With
```

```
End With
```

```
.Refresh
```

```
End With
```

and we get:



while following sample displays the item's drop down at its default position (no ShowPopupOffset call, default):

```
With Ribbon1
```

```
  .PopupAppearance = ShadowBorder
```

```
With .Items
```

```
  With .Add("Popup <b>1",2,100)
```

```
    .ShowPopupArrow = False
```

```
    .ShowPopupAlign = exShowPopupAlignDown
```

```
.ShowPopupOffset = "0,0:control"
```

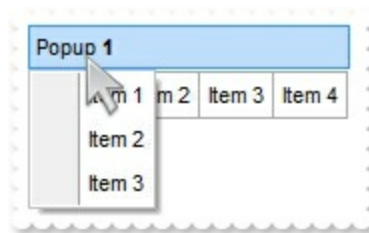


```

With .Items
    .Add "Item 1"
    .Add "Item 2"
    .Add "Item 3"
End With
End With
With .Add("Popup <b>2",2,200)
    .GroupPopup = exGroupPopup
    With .Items
        .Add "Item 2"
        .Add "Item 3"
        .Add "Item 4"
    End With
End With
End With
End With
.Refresh
End With

```

and we get:



How can I display the drop down popup to a different position?

VBA (MS Access, Excell...)

```

With Ribbon1
    With .Items
        With .Add("Popup",2,100)
            .ShowPopupAlign = 1
            .ShowPopupArrow = False
            .ShowPopupOffset = "-12,-48"
        End With
        With .Items
            .PopupAppearance = 3
            .Add "Item 1"
        End With
    End With
End With

```

```
.Add "Item 2"  
.Add "Item 3"  
End With  
End With  
End With  
.Refresh  
End With
```

VB6

```
With Ribbon1  
  With .Items  
    With .Add("Popup",2,100)  
      .ShowPopupAlign = exShowPopupAlignDown  
      .ShowPopupArrow = False  
      .ShowPopupOffset = "-12,-48"  
    With .Items  
      .PopupAppearance = RaisedBorder  
      .Add "Item 1"  
      .Add "Item 2"  
      .Add "Item 3"  
    End With  
  End With  
End With  
End With  
.Refresh  
End With
```

VB.NET

```
With Exribbon1  
  With .Items  
    With .Add("Popup",2,100)  
      .ShowPopupAlign =  
exontrol.EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown  
      .ShowPopupArrow = False  
      .ShowPopupOffset = "-12,-48"  
    With .Items  
      .PopupAppearance = exontrol.EXRIBBONLib.AppearanceEnum.RaisedBorder
```

```
.Add("Item 1")
.Add("Item 2")
.Add("Item 3")
End With
End With
End With
.Refresh()
End With
```

VB.NET for /COM

```
With AxRibbon1
  With .Items
    With .Add("Popup",2,100)
      .ShowPopupAlign =
EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown
      .ShowPopupArrow = False
      .ShowPopupOffset = "-12,-48"
    With .Items
      .PopupAppearance = EXRIBBONLib.AppearanceEnum.RaisedBorder
      .Add("Item 1")
      .Add("Item 2")
      .Add("Item 3")
    End With
  End With
End With
.Refresh()
End With
```

C++

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

#import <ExRibbon.dll>
using namespace EXRIBBONLib;
```

*/

```
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)->GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Popup",long(2),long(100));
    var_Item->PutShowPopupAlign(EXRIBBONLib::exShowPopupAlignDown);
    var_Item->PutShowPopupArrow(VARIANT_FALSE);
    var_Item->PutShowPopupOffset(L"-12,-48");
    EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();
    var_Items1->PutPopupAppearance(EXRIBBONLib::RaisedBorder);
    var_Items1->Add(L"Item 1",vtMissing,vtMissing);
    var_Items1->Add(L"Item 2",vtMissing,vtMissing);
    var_Items1->Add(L"Item 3",vtMissing,vtMissing);
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    Exribbonlib_tlb::IItemPtr var_Item = var_Items-
>Add(L"Popup",TVariant(2),TVariant(100));
    var_Item->ShowPopupAlign =
Exribbonlib_tlb::ShowPopupAlignEnum::exShowPopupAlignDown;
    var_Item->ShowPopupArrow = false;
    var_Item->ShowPopupOffset = L"-12,-48";
    Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;
    var_Items1->PopupAppearance =
Exribbonlib_tlb::AppearanceEnum::RaisedBorder;
    var_Items1->Add(L"Item 1",TNoParam(),TNoParam());
    var_Items1->Add(L"Item 2",TNoParam(),TNoParam());
    var_Items1->Add(L"Item 3",TNoParam(),TNoParam());
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
    exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Popup",2,100);
```

```

var_Item.ShowPopupAlign =
exontrol.EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown;
var_Item.ShowPopupArrow = false;
var_Item.ShowPopupOffset = "-12,-48";
exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
var_Items1.PopupAppearance =
exontrol.EXRIBBONLib.AppearanceEnum.RaisedBorder;
var_Items1.Add("Item 1",null,null);
var_Items1.Add("Item 2",null,null);
var_Items1.Add("Item 3",null,null);
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
var var_Items = Ribbon1.Items;
var var_Item = var_Items.Add("Popup",2,100);
var_Item.ShowPopupAlign = 1;
var_Item.ShowPopupArrow = false;
var_Item.ShowPopupOffset = "-12,-48";
var var_Items1 = var_Item.Items;
var_Items1.PopupAppearance = 3;
var_Items1.Add("Item 1",null,null);
var_Items1.Add("Item 2",null,null);
var_Items1.Add("Item 3",null,null);
Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Ribbon1
    With .Items
      With .Add("Popup",2,100)
        .ShowPopupAlign = 1
        .ShowPopupArrow = False
        .ShowPopupOffset = "-12,-48"
      With .Items
        .PopupAppearance = 3
        .Add "Item 1"
        .Add "Item 2"
        .Add "Item 3"
      End With
    End With
  End With
  .Refresh
End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;
EXRIBBONLib.Item var_Item = var_Items.Add("Popup",2,100);
var_Item.ShowPopupAlign =
EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown;
var_Item.ShowPopupArrow = false;
var_Item.ShowPopupOffset = "-12,-48";
EXRIBBONLib.Items var_Items1 = var_Item.Items;
```

```

var_Items1.PopupAppearance = EXRIBBONLib.AppearanceEnum.RaisedBorder;
var_Items1.Add("Item 1",null,null);
var_Items1.Add("Item 2",null,null);
var_Items1.Add("Item 3",null,null);
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Items,com_Items1;
    anytype var_Item,var_Items,var_Items1;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item =
com_Items.Add("Popup",COMVariant::createFromInt(2),COMVariant::createFromInt(100
com_Item = var_Item;
    com_Item.ShowPopupAlign(1/*exShowPopupAlignDown*/);
    com_Item.ShowPopupArrow(false);
    com_Item.ShowPopupOffset("-12,-48");
    var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
    com_Items1.PopupAppearance(3/*RaisedBorder*/);
    com_Items1.Add("Item 1");
    com_Items1.Add("Item 2");
    com_Items1.Add("Item 3");
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin

```

```

with Add('Popup',TObject(2),TObject(100)) do
begin
    ShowPopupAlign :=
EXRIBBONLib.ShowPopupAlignEnum.exShowPopupAlignDown;
    ShowPopupArrow := False;
    ShowPopupOffset := '-12,-48';
    with Items do
    begin
        PopupAppearance := EXRIBBONLib.AppearanceEnum.RaisedBorder;
        Add('Item 1',Nil,Nil);
        Add('Item 2',Nil,Nil);
        Add('Item 3',Nil,Nil);
    end;
end;
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items do
    begin
        with Add('Popup',OleVariant(2),OleVariant(100)) do
        begin
            ShowPopupAlign := EXRIBBONLib_TLB.exShowPopupAlignDown;
            ShowPopupArrow := False;
            ShowPopupOffset := '-12,-48';
            with Items do
            begin
                PopupAppearance := EXRIBBONLib_TLB.RaisedBorder;
                Add('Item 1',Null,Null);
                Add('Item 2',Null,Null);
                Add('Item 3',Null,Null);
            end;
        end;
    end;
end;
end;

```



```
end;  
Refresh();  
end
```

VFP

```
with thisform.Ribbon1  
  with .Items  
    with .Add("Popup",2,100)  
      .ShowPopupAlign = 1  
      .ShowPopupArrow = .F.  
      .ShowPopupOffset = "-12,-48"  
    with .Items  
      .PopupAppearance = 3  
      .Add("Item 1")  
      .Add("Item 2")  
      .Add("Item 3")  
    endwith  
  endwith  
endwith  
.Refresh  
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Items,var_Items1  
  
oRibbon = form.Activex1.nativeObject  
var_Items = oRibbon.Items  
var_Item = var_Items.Add("Popup",2,100)  
var_Item.ShowPopupAlign = 1  
var_Item.ShowPopupArrow = false  
var_Item.ShowPopupOffset = "-12,-48"  
var_Items1 = var_Item.Items  
var_Items1.PopupAppearance = 3  
var_Items1.Add("Item 1")  
var_Items1.Add("Item 2")  
var_Items1.Add("Item 3")
```

```
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
Dim var_Item as P
Dim var_Items as P
Dim var_Items1 as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
  var_Item = var_Items.Add("Popup",2,100)
    var_Item.ShowPopupAlign = 1
    var_Item.ShowPopupArrow = .f.
    var_Item.ShowPopupOffset = "-12,-48"
    var_Items1 = var_Item.Items
      var_Items1.PopupAppearance = 3
      var_Items1.Add("Item 1")
      var_Items1.Add("Item 2")
      var_Items1.Add("Item 3")
oRibbon.Refresh()
```

Visual Objects

```
local var_Item as IItem
local var_Items,var_Items1 as IItems

var_Items := oDCOCX_Exontrol1.Items
  var_Item := var_Items.Add("Popup",2,100)
    var_Item.ShowPopupAlign := exShowPopupAlignDown
    var_Item.ShowPopupArrow := false
    var_Item.ShowPopupOffset := "-12,-48"
    var_Items1 := var_Item.Items
      var_Items1.PopupAppearance := RaisedBorder
      var_Items1.Add("Item 1",nil,nil)
      var_Items1.Add("Item 2",nil,nil)
```

```
var_Items1:Add("Item 3",nil,nil)
oDCOCX_Exontrol1:Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Items,var_Items1
```

```
oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Item = var_Items.Add("Popup",2,100)
var_Item.ShowPopupAlign = 1
var_Item.ShowPopupArrow = false
var_Item.ShowPopupOffset = "-12,-48"
var_Items1 = var_Item.Items
var_Items1.PopupAppearance = 3
var_Items1.Add("Item 1")
var_Items1.Add("Item 2")
var_Items1.Add("Item 3")
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
Forward Send OnCreate
Variant voltems
Get ComItems to voltems
Handle holtems
Get Create (RefClass(cComItems)) to holtems
Set pvComObject of holtems to voltems
Variant voltem
Get ComAdd of holtems "Popup" 2 100 to voltem
Handle holtem
Get Create (RefClass(cComItem)) to holtem
Set pvComObject of holtem to voltem
Set ComShowPopupAlign of holtem to OLEexShowPopupAlignDown
Set ComShowPopupArrow of holtem to False
```

Set **ComShowPopupOffset** of holtem to "-12,-48"

Variant voltems1

Get ComItems of holtem to voltems1

Handle holtems1

Get Create (RefClass(cComItems)) to holtems1

Set pvComObject of holtems1 to voltems1

Set ComPopupAppearance of holtems1 to OLERaisedBorder

Get ComAdd of holtems1 "Item 1" Nothing Nothing to Nothing

Get ComAdd of holtems1 "Item 2" Nothing Nothing to Nothing

Get ComAdd of holtems1 "Item 3" Nothing Nothing to Nothing

Send Destroy to holtems1

Send Destroy to holtem

Send Destroy to holtems

Send ComRefresh

End_Procedure

XBase++

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
    LOCAL oForm
```

```
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
    LOCAL oltem
```

```
    LOCAL oltems,oltems1
```

```
    LOCAL oRibbon
```

```
    oForm := XbpDialog():new( AppDesktop() )
```

```
    oForm:drawingArea:clipChildren := .T.
```

```
    oForm:create( „{100,100}, {640,480},„ .F. )
```

```
    oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
```

```
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-  
CFBE431702E2}*/
```

```
    oRibbon:create(„ {10,60},{610,370} )
```

```
oItems := oRibbon:Items()
oItem := oItems:Add("Popup",2,100)
oItem>ShowPopupAlign := 1/*exShowPopupAlignDown*/
oItem>ShowPopupArrow := .F.
oItem:ShowPopupOffset := "-12,-48"
oItems1 := oItem:Items()
oItems1:PopupAppearance := 3/*RaisedBorder*/
oItems1:Add("Item 1")
oItems1:Add("Item 2")
oItems1:Add("Item 3")
oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN
```

property Item.ShowPopupOnChecked as Boolean

Specifies whether the item's sub menu is shown only if the item is checked.

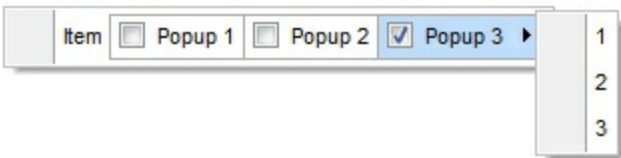
Type	Description
Boolean	A Boolean expression that specifies whether the item's sub menu is shown only if the item is checked.

By default, the ShowPopupOnChecked property is False. The [Check](#) property assigns a check box to the current item. The [SubMenu](#) property specifies the sub-items of the current item. The ShowPopupOnChecked property has effect only if the item displays sub items (the SubMenu.Count property is not zero). The [ShowCheckedAsSelected](#) property specifies how the checked item is displayed.

The following screen shot show items with ShowPopupOnChecked on False (default) (*please notice that all items display the popup-arrow*):



The following screen shot show items with ShowPopupOnChecked on True (*please notice that just checked popup displays the popup-arrow, and so the sub-menu*):



Is it possible to expand an item when it is clicked (tree,group,vertical)?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("Expand",2)
      .GroupPopup = 259 ' GroupPopupEnum.exGroupPopupVertical Or
GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
      .Check = True
      .ShowPopupOnChecked = True
    With .Items
      .Padding = "22,0,0,0"
```

```

.Add("Radio 1").Radio = True
.Add("Radio 2").Radio = True
With .Add("Radio 3")
    .Radio = True
    .Checked = True
End With
End With
.Checked = True
End With
End With
.Refresh
End With

```

VB6

```

With Ribbon1
    With .Items
        With .Add("Expand",2)
            .GroupPopup = GroupPopupEnum.exGroupPopupVertical Or
GroupPopupEnum.exNoGroupPopupFrame Or GroupPopupEnum.exGroupPopup
            .Check = True
            .ShowPopupOnChecked = True
        End With
    End With
    .Padding = "22,0,0,0"
    .Add("Radio 1").Radio = True
    .Add("Radio 2").Radio = True
    With .Add("Radio 3")
        .Radio = True
        .Checked = True
    End With
End With
.Checked = True
End With
.Refresh
End With

```

VB.NET

With Exribbon1

With .Items

With .Add("Expand",2)

.GroupPopup =

exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopupVertical Or

exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or

exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup

.Check = True

.ShowPopupOnChecked = True

With .Items

.Padding = "22,0,0,0"

.Add("Radio 1").Radio = True

.Add("Radio 2").Radio = True

With .Add("Radio 3")

.Radio = True

.Checked = True

End With

End With

.Checked = True

End With

End With

.Refresh()

End With

VB.NET for /COM

With AxRibbon1

With .Items

With .Add("Expand",2)

.GroupPopup = EXRIBBONLib.GroupPopupEnum.exGroupPopupVertical Or

EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or

EXRIBBONLib.GroupPopupEnum.exGroupPopup

.Check = True

.ShowPopupOnChecked = True

With .Items

.Padding = "22,0,0,0"

.Add("Radio 1").Radio = True


```

        .Add("Radio 2").Radio = True
    With .Add("Radio 3")
        .Radio = True
        .Checked = True
    End With
End With
.Checked = True
End With
End With
.Refresh()
End With

```

C++

```

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
> GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Expand",long(2),vtMissing);
    var_Item->
> PutGroupPopup(EXRIBBONLib::GroupPopupEnum(EXRIBBONLib::exGroupPopupVert
| EXRIBBONLib::exNoGroupPopupFrame | EXRIBBONLib::exGroupPopup));
    var_Item->PutCheck(VARIANT_TRUE);
    var_Item->PutShowPopupOnChecked(VARIANT_TRUE);
    EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();
    var_Items1->PutPadding(L"22,0,0,0");
    var_Items1->Add(L"Radio 1",vtMissing,vtMissing)->PutRadio(VARIANT_TRUE);
    var_Items1->Add(L"Radio 2",vtMissing,vtMissing)->PutRadio(VARIANT_TRUE);
    EXRIBBONLib::IItemPtr var_Item1 = var_Items1->Add(L"Radio
3",vtMissing,vtMissing);

```

```

var_Item1->PutRadio(VARIANT_TRUE);
var_Item1->PutChecked(VARIANT_TRUE);
var_Item->PutChecked(VARIANT_TRUE);
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
Exribbonlib_tlb::IItemPtr var_Item = var_Items-
>Add(L"Expand",TVariant(2),TNoParam());
var_Item->GroupPopup =
Exribbonlib_tlb::GroupPopupEnum::exGroupPopupVertical |
Exribbonlib_tlb::GroupPopupEnum::exNoGroupPopupFrame |
Exribbonlib_tlb::GroupPopupEnum::exGroupPopup;
var_Item->Check = true;
var_Item->ShowPopupOnChecked = true;
Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;
var_Items1->Padding = L"22,0,0,0";
var_Items1->Add(L"Radio 1",TNoParam(),TNoParam())->Radio = true;
var_Items1->Add(L"Radio 2",TNoParam(),TNoParam())->Radio = true;
Exribbonlib_tlb::IItemPtr var_Item1 = var_Items1->Add(L"Radio
3",TNoParam(),TNoParam());
var_Item1->Radio = true;
var_Item1->Checked = true;
var_Item->Checked = true;
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Expand",2,null);
var_Item.GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopupVertical |
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup;
var_Item.Check = true;

```

```

var_Item.ShowPopupOnChecked = true;
exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
var_Items1.Padding = "22,0,0,0";
var_Items1.Add("Radio 1",null,null).Radio = true;
var_Items1.Add("Radio 2",null,null).Radio = true;
exontrol.EXRIBBONLib.Item var_Item1 = var_Items1.Add("Radio 3",null,null);
var_Item1.Radio = true;
var_Item1.Checked = true;
var_Item.Checked = true;
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
var var_Items = Ribbon1.Items;
var var_Item = var_Items.Add("Expand",2,null);
var_Item.GroupPopup = 259;
var_Item.Check = true;
var_Item.ShowPopupOnChecked = true;
var var_Items1 = var_Item.Items;
var_Items1.Padding = "22,0,0,0";
var_Items1.Add("Radio 1",null,null).Radio = true;
var_Items1.Add("Radio 2",null,null).Radio = true;
var var_Item1 = var_Items1.Add("Radio 3",null,null);
var_Item1.Radio = true;
var_Item1.Checked = true;
var_Item.Checked = true;
Ribbon1.Refresh();
}
</SCRIPT>

```

</BODY>

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Ribbon1
    With .Items
      With .Add("Expand",2)
        .GroupPopup = 259 ' GroupPopupEnum.exGroupPopupVertical Or
GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
        .Check = True
        .ShowPopupOnChecked = True
      With .Items
        .Padding = "22,0,0,0"
        .Add("Radio 1").Radio = True
        .Add("Radio 2").Radio = True
        With .Add("Radio 3")
          .Radio = True
          .Checked = True
        End With
      End With
    End With
    .Checked = True
  End With
  .Refresh
End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;
    EXRIBBONLib.Item var_Item = var_Items.Add("Expand",2,null);
        var_Item.GroupPopup = EXRIBBONLib.GroupPopupEnum.exGroupPopupVertical
| EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
EXRIBBONLib.GroupPopupEnum.exGroupPopup;
    var_Item.Check = true;
    var_Item.ShowPopupOnChecked = true;
    EXRIBBONLib.Items var_Items1 = var_Item.Items;
        var_Items1.Padding = "22,0,0,0";
        var_Items1.Add("Radio 1",null,null).Radio = true;
        var_Items1.Add("Radio 2",null,null).Radio = true;
        EXRIBBONLib.Item var_Item1 = var_Items1.Add("Radio 3",null,null);
            var_Item1.Radio = true;
            var_Item1.Checked = true;
        var_Item.Checked = true;
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Item,com_Item1,com_Items,com_Items1;
    anytype var_Item,var_Item1,var_Items,var_Items1;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
        var_Item = com_Items.Add("Expand",COMVariant::createFromInt(2)); com_Item =
var_Item;
            com_Item.GroupPopup(259/*exGroupPopupVertical |
exNoGroupPopupFrame | exGroupPopup*/);
            com_Item.Check(true);
            com_Item.ShowPopupOnChecked(true);
            var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
```

```

        com_Items1.Padding("22,0,0,0");
        var_Item1 = COM::createFromObject(com_Items1.Add("Radio 1"));
com_Item1 = var_Item1;
        com_Item1.Radio(true);
        var_Item1 = COM::createFromObject(com_Items1.Add("Radio 2"));
com_Item1 = var_Item1;
        com_Item1.Radio(true);
        var_Item1 = com_Items1.Add("Radio 3"); com_Item1 = var_Item1;
        com_Item1.Radio(true);
        com_Item1.Checked(true);
        com_Item.Checked(true);
        exribbon1.Refresh();
    }

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items do
    begin
        with Add('Expand',TObject(2),Nil) do
        begin
            GroupPopup :=
Integer(EXRIBBONLib.GroupPopupEnum.exGroupPopupVertical) Or
Integer(EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib.GroupPopupEnum.exGroupPopup);
            Check := True;
            ShowPopupOnChecked := True;
            with Items do
            begin
                Padding := '22,0,0,0';
                Add('Radio 1',Nil,Nil).Radio := True;
                Add('Radio 2',Nil,Nil).Radio := True;
                with Add('Radio 3',Nil,Nil) do
                begin
                    Radio := True;
                    Checked := True;

```

```
        end;
    end;
    Checked := True;
end;
end;
Refresh();
end
```

Delphi (standard)

```
with Ribbon1 do
begin
    with Items do
    begin
        with Add('Expand',OleVariant(2),Null) do
        begin
            GroupPopup := Integer(EXRIBBONLib_TLB.exGroupPopupVertical) Or
Integer(EXRIBBONLib_TLB.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib_TLB.exGroupPopup);
            Check := True;
            ShowPopupOnChecked := True;
            with Items do
            begin
                Padding := '22,0,0,0';
                Add('Radio 1',Null,Null).Radio := True;
                Add('Radio 2',Null,Null).Radio := True;
                with Add('Radio 3',Null,Null) do
                begin
                    Radio := True;
                    Checked := True;
                end;
            end;
            Checked := True;
        end;
    end;
    Refresh();
end
```

```

with thisform.Ribbon1
  with .Items
    with .Add("Expand",2)
      .GroupPopup = 259 && GroupPopupEnum.exGroupPopupVertical Or
GroupPopupEnum.exNoGroupPopupFrame Or GroupPopupEnum.exGroupPopup
      .Check = .T.
      .ShowPopupOnChecked = .T.
    with .Items
      .Padding = "22,0,0,0"
      .Add("Radio 1").Radio = .T.
      .Add("Radio 2").Radio = .T.
      with .Add("Radio 3")
        .Radio = .T.
        .Checked = .T.
      endwhile
    endwhile
    .Checked = .T.
  endwhile
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Item2,var_Item3,var_Items,var_Items1

oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("Expand",2)
var_Item.GroupPopup = 259 /*exGroupPopupVertical | exNoGroupPopupFrame
| exGroupPopup*/
var_Item.Check = true
var_Item.ShowPopupOnChecked = true
var_Items1 = var_Item.Items
var_Items1.Padding = "22,0,0,0"
// var_Items1.Add("Radio 1").Radio = true

```



```

var_Item1 = var_Items1.Add("Radio 1")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.Radio = true]
endwith
// var_Items1.Add("Radio 2").Radio = true
var_Item2 = var_Items1.Add("Radio 2")
with (oRibbon)
    TemplateDef = [Dim var_Item2]
    TemplateDef = var_Item2
    Template = [var_Item2.Radio = true]
endwith
var_Item3 = var_Items1.Add("Radio 3")
var_Item3.Radio = true
var_Item3.Checked = true
var_Item.Checked = true
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Item2 as P
Dim var_Item3 as P
Dim var_Items as P
Dim var_Items1 as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Item = var_Items.Add("Expand",2)
var_Item.GroupPopup = 259 'exGroupPopupVertical +
exNoGroupPopupFrame + exGroupPopup
var_Item.Check = .t.
var_Item.ShowPopupOnChecked = .t.

```

```

var_Items1 = var_Item.Items
var_Items1.Padding = "22,0,0,0"
' var_Items1.Add("Radio 1").Radio = .t.
var_Item1 = var_Items1.Add("Radio 1")
oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1
oRibbon.Template = "var_Item1.Radio = True"

' var_Items1.Add("Radio 2").Radio = .t.
var_Item2 = var_Items1.Add("Radio 2")
oRibbon.TemplateDef = "Dim var_Item2"
oRibbon.TemplateDef = var_Item2
oRibbon.Template = "var_Item2.Radio = True"

var_Item3 = var_Items1.Add("Radio 3")
var_Item3.Radio = .t.
var_Item3.Checked = .t.
var_Item.Checked = .t.
oRibbon.Refresh()

```

Visual Objects

```

local var_Item,var_Item1 as Item
local var_Items,var_Items1 as Items

var_Items := oDCOCX_Exontrol1.Items
var_Item := var_Items.Add("Expand",2,nil)
var_Item:GroupPopup := exGroupPopupVertical | exNoGroupPopupFrame |
exGroupPopup
var_Item:Check := true
var_Item:ShowPopupOnChecked := true
var_Items1 := var_Item.Items
var_Items1.Padding := "22,0,0,0"
var_Items1.Add("Radio 1",nil,nil):Radio := true
var_Items1.Add("Radio 2",nil,nil):Radio := true
var_Item1 := var_Items1.Add("Radio 3",nil,nil)

```

```
var_Item1:Radio := true
var_Item1:Checked := true
var_Item:Checked := true
oDCOCX_Exontrol1:Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Item1,var_Items,var_Items1

oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Item = var_Items.Add("Expand",2)
var_Item.GroupPopup = 259 /*exGroupPopupVertical | exNoGroupPopupFrame
| exGroupPopup*/
var_Item.Check = true
var_Item.ShowPopupOnChecked = true
var_Items1 = var_Item.Items
var_Items1.Padding = "22,0,0,0"
var_Items1.Add("Radio 1").Radio = true
var_Items1.Add("Radio 2").Radio = true
var_Item1 = var_Items1.Add("Radio 3")
var_Item1.Radio = true
var_Item1.Checked = true
var_Item.Checked = true
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
Forward Send OnCreate
Variant voltems
Get ComItems to voltems
Handle holtems
Get Create (RefClass(cComItems)) to holtems
Set pvComObject of holtems to voltems
Variant voltem
```

Get ComAdd of holtems "Expand" 2 Nothing to voltem
Handle holtem
Get Create (RefClass(cComItem)) to holtem
Set pvComObject of holtem to voltem
Set **ComGroupPopup** of holtem to (OLEexGroupPopupVertical +
OLEexNoGroupPopupFrame + OLEexGroupPopup)
Set **ComCheck** of holtem to True
Set **ComShowPopupOnChecked** of holtem to True
Variant voltems1
Get ComItems of holtem to voltems1
Handle holtems1
Get Create (RefClass(cComItems)) to holtems1
Set pvComObject of holtems1 to voltems1
Set ComPadding of holtems1 to "22,0,0,0"
Variant voltem1
Get ComAdd of holtems1 "Radio 1" Nothing Nothing to voltem1
Handle holtem1
Get Create (RefClass(cComItem)) to holtem1
Set pvComObject of holtem1 to voltem1
Set ComRadio of holtem1 to True
Send Destroy to holtem1
Variant voltem2
Get ComAdd of holtems1 "Radio 2" Nothing Nothing to voltem2
Handle holtem2
Get Create (RefClass(cComItem)) to holtem2
Set pvComObject of holtem2 to voltem2
Set ComRadio of holtem2 to True
Send Destroy to holtem2
Variant voltem3
Get ComAdd of holtems1 "Radio 3" Nothing Nothing to voltem3
Handle holtem3
Get Create (RefClass(cComItem)) to holtem3
Set pvComObject of holtem3 to voltem3
Set ComRadio of holtem3 to True
Set ComChecked of holtem3 to True
Send Destroy to holtem3
Send Destroy to holtems1

```
Set ComChecked of holtem to True
Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem,oltem1
    LOCAL oltems,oltems1
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oltems := oRibbon:Items()
    oltem := oltems:Add("Expand",2)
    oltem:GroupPopup :=
259/*exGroupPopupVertical+exNoGroupPopupFrame+exGroupPopup*/
    oltem:Check := .T.
    oltem:ShowPopupOnChecked := .T.
    oltems1 := oltem:Items()
    oltems1:Padding := "22,0,0,0"
    oltems1:Add("Radio 1"):Radio := .T.
```

```
    oltems1:Add("Radio 2"):Radio := .T.  
    oltem1 := oltems1:Add("Radio 3")  
        oltem1:Radio := .T.  
        oltem1:Checked := .T.  
    oltem:Checked := .T.  
oRibbon:Refresh()  
  
oForm:Show()  
DO WHILE nEvent != xbeP_Quit  
    nEvent := AppEvent( @mp1, @mp2, @oXbp )  
    oXbp:handleEvent( nEvent, mp1, mp2 )  
ENDDO  
RETURN
```

property Item.Strikeout as Boolean

Specifies whether the item's caption should appear in strikeout.

Type	Description
Boolean	A Boolean expression that specifies whether the item's caption is shown in strikeout.

By default, the Strikeout property is False. Use the Strikeout property to show the item's caption in strikeout. The [Caption](#) property indicates the HTML caption to be shown on the item. The <s> HTML tag can be used on the item's Caption property to specify different parts of the caption to be shown in strikeout.

How can I show the item as strikeout?

VBA (MS Access, Excell...)

```
With Ribbon1
    With .Items
        .Add("Item").Strikeout = True
        .Add "<s> Item</s> "
        .Add("").ToString = "Item[stk]"
    End With
    .Refresh
End With
```

VB6

```
With Ribbon1
    With .Items
        .Add("Item").Strikeout = True
        .Add "<s> Item</s> "
        .Add("").ToString = "Item[stk]"
    End With
    .Refresh
End With
```

VB.NET

```
With Exribbon1
    With .Items
```

```

.Add("Item").Strikeout = True
.Add("<s>Item</s>")
.Add("").ToString = "Item[stk]"
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    With .Items
        .Add("Item").Strikeout = True
        .Add("<s>Item</s>")
        .Add("").ToString = "Item[stk]"
    End With
    .Refresh()
End With

```

C++

```

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
var_Items->Add(L"Item",vtMissing,vtMissing)->PutStrikeout(VARIANT_TRUE);
var_Items->Add(L"<s>Item</s>",vtMissing,vtMissing);
var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"Item[stk]");
spRibbon1->Refresh();

```

C++ Builder


```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
var_Items->Add(L"Item",TNoParam(),TNoParam())->Strikeout = true;
var_Items->Add(L"<s>Item</s>",TNoParam(),TNoParam());
var_Items->Add(L"",TNoParam(),TNoParam())->ToString = L"Item[stk]";
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
var_Items.Add("Item",null,null).Strikeout = true;
var_Items.Add("<s>Item</s>",null,null);
var_Items.Add("",null,null).ToString = "Item[stk]";
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var_Items.Add("Item",null,null).Strikeout = true;
    var_Items.Add("<s>Item</s>",null,null);
    var_Items.Add("",null,null).ToString = "Item[stk]";
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>

```

```

<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Ribbon1
    With .Items
      .Add("Item").Strikeout = True
      .Add "<s>Item</s>"
      .Add("").ToString = "Item[stk]"
    End With
    .Refresh
  End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.Add("Item",null,null).Strikeout = true;
var_Items.Add("<s>Item</s>",null,null);
var_Items.Add("",null,null).ToString = "Item[stk]";
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
  COM com_Item,com_Items;
  anytype var_Item,var_Items;
  ;

  super();

  var_Items = exribbon1.Items(); com_Items = var_Items;

```

```

    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
    com_Item.Strikeout(true);
    com_Items.Add("<s>Item</s>");
    var_Item = COM::createFromObject(com_Items.Add("")); com_Item = var_Item;
    com_Item.ToString("Item[stk]");
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
  with Items do
  begin
    Add('Item',Nil,Nil).Strikeout := True;
    Add('<s>Item</s>',Nil,Nil);
    Add('',Nil,Nil).ToString := 'Item[stk]';
  end;
  Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  with Items do
  begin
    Add('Item',Null,Null).Strikeout := True;
    Add('<s>Item</s>',Null,Null);
    Add('',Null,Null).ToString := 'Item[stk]';
  end;
  Refresh();
end

```

VFP

```

with thisform.Ribbon1

```

```

with .Items
    .Add("Item").Strikeout = .T.
    .Add("<s>Item</s> ")
    .Add("").ToString = "Item[stk]"
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Items

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
// var_Items.Add("Item").Strikeout = true
var_Item = var_Items.Add("Item")
with (oRibbon)
    TemplateDef = [Dim var_Item]
    TemplateDef = var_Item
    Template = [var_Item.Strikeout = true]
endwith
var_Items.Add("<s>Item</s> ")
// var_Items.Add("").ToString = "Item[stk]"
var_Item1 = var_Items.Add("")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.ToString = "Item[stk]"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P

```

```

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
' var_Items.Add("Item").Strikeout = .t.
var_Item = var_Items.Add("Item")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.Strikeout = True"

var_Items.Add("<s>Item</s>")
' var_Items.Add("").ToString = "Item[stk]"
var_Item1 = var_Items.Add("")
oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1
oRibbon.Template = "var_Item1.ToString = \"Item[stk]\""

oRibbon.Refresh()

```

Visual Objects

```

local var_Items as IItems

var_Items := oDCOCX_Exontrol1.Items
var_Items.Add("Item",nil,nil):Strikeout := true
var_Items.Add("<s>Item</s>",nil,nil)
var_Items.Add("",nil,nil):ToString := "Item[stk]"
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```

OleObject oRibbon,var_Items

oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Items.Add("Item").Strikeout = true
var_Items.Add("<s>Item</s>")

```

```
var_Items.Add("").ToString = "Item[stk]"  
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate  
    Forward Send OnCreate  
    Variant voltems  
    Get ComItems to voltems  
    Handle holtems  
    Get Create (RefClass(cComItems)) to holtems  
    Set pvComObject of holtems to voltems  
        Variant voltem  
        Get ComAdd of holtems "Item" Nothing Nothing to voltem  
        Handle holtem  
        Get Create (RefClass(cComItem)) to holtem  
        Set pvComObject of holtem to voltem  
            Set ComStrikeout of holtem to True  
        Send Destroy to holtem  
        Get ComAdd of holtems "<s>Item</s>" Nothing Nothing to Nothing  
        Variant voltem1  
        Get ComAdd of holtems "" Nothing Nothing to voltem1  
        Handle holtem1  
        Get Create (RefClass(cComItem)) to holtem1  
        Set pvComObject of holtem1 to voltem1  
            Set ComToString of holtem1 to "Item[stk]"  
        Send Destroy to holtem1  
    Send Destroy to holtems  
    Send ComRefresh  
End_Procedure
```

XBase++

```
#include "AppEvent.ch"  
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

LOCAL oForm

LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

LOCAL oltems

LOCAL oRibbon

oForm := XbpDialog():new(AppDesktop())

oForm:drawingArea:clipChildren := .T.

oForm:create(„{100,100}, {640,480}„, .F.)

oForm:close := {|| PostAppEvent(xbeP_Quit)}

oRibbon := XbpActiveXControl():new(oForm:drawingArea)

oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-CFBE431702E2}*/

oRibbon:create(„ {10,60},{610,370})

oltems := oRibbon:Items()

oltems:Add("Item"):Strikeout := .T.

oltems:Add("<s>Item</s>")

oltems:Add(""):ToString := "Item[stk]"

oRibbon:Refresh()

oForm:Show()

DO WHILE nEvent != xbeP_Quit

nEvent := AppEvent(@mp1, @mp2, @oXbp)

oXbp:handleEvent(nEvent, mp1, mp2)

ENDDO

RETURN

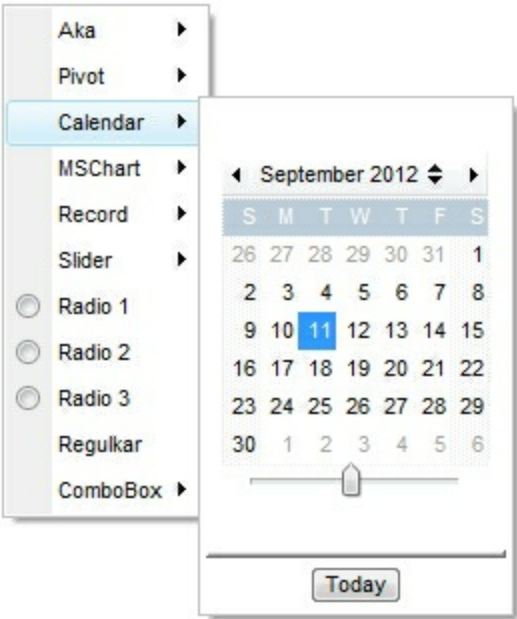
property Item.SubControl as Control

Retrieves the Control object that holds information about item's inside component.

Type	Description
Control	A Control object that holds properties to handle the inside ActiveX item.

Use the SubControl property when using the [ItemTypeEnum.SubControl](#) to add an item that hosts an ActiveX inside. Use the [ControlID](#) property to specify the IDentifier of the object to be displayed on the item. Use the [Create](#) method to create an inside ActiveX control. The inside ActiveX control fires the events through the [OleEvent](#) event.

The following screen shot displays an item with an [ExCalendar](#) inside:



The following samples shows how to load an ActiveX control ([Exontrol.Calendar](#))

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items.Add("Calendar",3).SubControl
    .Width = 256
    .Height = 256
    .ControlID = "Exontrol.Calendar"
    .Create
  End With
  .Refresh
End With
```


VB6

With Ribbon1

With .Items.Add("Calendar",3).SubControl

.Width = 256

.Height = 256

.ControlID = "Exontrol.Calendar"

.Create

End With

.Refresh

End With

VB.NET

With Exribbon1

With .Items.Add("Calendar",3).SubControl

.Width = 256

.Height = 256

.ControlID = "Exontrol.Calendar"

.Create()

End With

.Refresh()

End With

VB.NET for /COM

With AxRibbon1

With .Items.Add("Calendar",3).SubControl

.Width = 256

.Height = 256

.ControlID = "Exontrol.Calendar"

.Create()

End With

.Refresh()

End With

C++

/*

Copy and paste the following directives to your header file as it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control Library'

```
#import <ExRibbon.dll>
using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IControlPtr var_Control = spRibbon1->GetItems()-
>Add(L"Calendar",long(3),vtMissing)->GetSubControl();
var_Control->PutWidth(256);
var_Control->PutHeight(256);
var_Control->PutControlID(L"Exontrol.Calendar");
var_Control->Create();
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IControlPtr var_Control = Ribbon1->Items-
>Add(L"Calendar",TVariant(3),TNoParam())->SubControl;
var_Control->Width = 256;
var_Control->Height = 256;
var_Control->ControlID = L"Exontrol.Calendar";
var_Control->Create();
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Control var_Control =
exribbon1.Items.Add("Calendar",3,null).SubControl;
var_Control.Width = 256;
var_Control.Height = 256;
var_Control.ControlID = "Exontrol.Calendar";
var_Control.Create();
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Control = Ribbon1.Items.Add("Calendar",3,null).SubControl;
    var_Control.Width = 256;
    var_Control.Height = 256;
    var_Control.ControlID = "Exontrol.Calendar";
    var_Control.Create();
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items.Add("Calendar",3).SubControl
            .Width = 256
            .Height = 256
            .ControlID = "Exontrol.Calendar"
            .Create
        End With
    End With
    .Refresh
End Function
</SCRIPT>
</BODY>
```

```
End With  
End Function  
</SCRIPT>  
</BODY>
```

C# for /COM

```
EXRIBBONLib.Control var_Control =  
axRibbon1.Items.Add("Calendar",3,null).SubControl;  
    var_Control.Width = 256;  
    var_Control.Height = 256;  
    var_Control.ControlID = "Exontrol.Calendar";  
    var_Control.Create();  
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()  
{  
    COM com_Control,com_Item;  
    anytype var_Control,var_Item;  
    ;  
  
    super();  
  
    var_Item =  
COM::createFromObject(exribbon1.Items()).Add("Calendar",COMVariant::createFromInt  
com_Item = var_Item;  
    var_Control = com_Item.SubControl(); com_Control = var_Control;  
    com_Control.Width(256);  
    com_Control.Height(256);  
    com_Control.ControlID("Exontrol.Calendar");  
    com_Control.Create();  
    exribbon1.Refresh();  
}
```

Delphi 8 (.NET only)

```
with AxRibbon1 do
begin
  with Items.Add('Calendar',TObject(3),Nil).SubControl do
  begin
    Width := 256;
    Height := 256;
    ControlID := 'Exontrol.Calendar';
    Create();
  end;
  Refresh();
end
```

Delphi (standard)

```
with Ribbon1 do
begin
  with Items.Add('Calendar',OleVariant(3),Null).SubControl do
  begin
    Width := 256;
    Height := 256;
    ControlID := 'Exontrol.Calendar';
    Create();
  end;
  Refresh();
end
```

VFP

```
with thisform.Ribbon1
  with .Items.Add("Calendar",3).SubControl
    .Width = 256
    .Height = 256
    .ControlID = "Exontrol.Calendar"
    .Create
  endwith
.Refresh
```

endwith

dBASE Plus

```
local oRibbon,var_Control

oRibbon = form.Activex1.nativeObject
var_Control = oRibbon.Items.Add("Calendar",3).SubControl
    var_Control.Width = 256
    var_Control.Height = 256
    var_Control.ControlID = "Exontrol.Calendar"
    var_Control.Create()
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
Dim var_Control as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Control = oRibbon.Items.Add("Calendar",3).SubControl
    var_Control.Width = 256
    var_Control.Height = 256
    var_Control.ControlID = "Exontrol.Calendar"
    var_Control.Create()
oRibbon.Refresh()
```

Visual Objects

```
local var_Control as IControl

var_Control := oDCOCX_Exontrol1:Items:Add("Calendar",3,nil):SubControl
    var_Control:Width := 256
    var_Control:Height := 256
    var_Control:ControlID := "Exontrol.Calendar"
    var_Control:Create()
```

```
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Control
```

```
oRibbon = ole_1.Object
```

```
var_Control = oRibbon.Items.Add("Calendar",3).SubControl
```

```
var_Control.Width = 256
```

```
var_Control.Height = 256
```

```
var_Control.ControlID = "Exontrol.Calendar"
```

```
var_Control.Create()
```

```
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
```

```
Forward Send OnCreate
```

```
Variant voltems
```

```
Get ComItems to voltems
```

```
Handle holtems
```

```
Get Create (RefClass(cComItems)) to holtems
```

```
Set pvComObject of holtems to voltems
```

```
Variant voltem
```

```
Get ComAdd of holtems "Calendar" 3 Nothing to voltem
```

```
Handle holtem
```

```
Get Create (RefClass(cComItem)) to holtem
```

```
Set pvComObject of holtem to voltem
```

```
Variant voControl
```

```
Get ComSubControl of holtem to voControl
```

```
Handle hoControl
```

```
Get Create (RefClass(cComControl)) to hoControl
```

```
Set pvComObject of hoControl to voControl
```

```
Set ComWidth of hoControl to 256
```

```
Set ComHeight of hoControl to 256
```

```
Set ComControlID of hoControl to "Exontrol.Calendar"
```

```
    Send ComCreate of hoControl
    Send Destroy to hoControl
    Send Destroy to holtem
    Send Destroy to holtems
    Send ComRefresh
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oControl
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oControl := oRibbon:Items():Add("Calendar",3):SubControl()
    oControl:Width := 256
    oControl:Height := 256
    oControl:ControlID := "Exontrol.Calendar"
    oControl:Create()
    oRibbon:Refresh()

    oForm:Show()
    DO WHILE nEvent != xbeP_Quit
```



```
nEvent := AppEvent( @mp1, @mp2, @oXbp )
oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN
```

property Item.SubMenu as Items

Retrieves an Items collection that indicates the item's sub menu. Retrieves Nothing, if the item contains no sub menu.

Type	Description
Items	An Items collection that holds the Item objects to be displayed in the sub-menu.

The [Items](#) and SubMenu properties are equivalents. Use the SubMenu property to access the Item objects in a sub-menu item. The [Parent](#) property of the Item object returns an empty object, if the item contains no parent. The Parent item property can be used to access the parent of the item, when it is contained by a sub-menu.

property Item.Tab as Long

Specifies the identifier of the item/tab where the current group popup is shown instead.

Type	Description
Long	A Long expression that specifies the identifier of the item where the grouping items of the current item is shown.

property Item.Tooltip as String

Specifies the item's tooltip.

Type	Description
String	A String expression that defines the HTML caption to be displayed when the cursor hovers the item.

The Tooltip property assigns a HTML tooltip to an item, that's displayed only when the cursor hovers the item. The [TooltipTitle](#) property specifies the title for the item's tooltip. The [TooltipDelay](#) property specifies the time until the tooltip is shown. Use the [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipWidth](#) property to specify the width of the tooltip window. The [ToolTipFont](#) property specifies the tooltip's font. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color.

The ToolTip property supports the following HTML tags:

- ** ... ** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** Strike-through text
- **<a id;options> ... ** displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- ** ... ** 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
- **`number[:width]`** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **`key[:width]`** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **`&`**; (&), **`<`**; (<), **`>`**; (>), **`"`**; (") and **`&#number;`**; (the character with specified code), For instance, the `€` displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a `#character` and a digit. For instance if you want to display `bold` in HTML caption you can use `bold`
- **`<off offset> ... </off>`** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated `</off>` tag is found. You can use the `<off offset>` HTML tag in combination with the `` to define a smaller or a larger font to be displayed. For instance: "Text with `<off 6>`subscript" displays the text such as: Text with subscript The "Text with `<off -6>`superscript" displays the text such as: Text with superscript
- **`<gra rrggbb;mode;blend> ... </gra>`** defines a gradient text. The text color or `<fgcolor>` defines the starting gradient color, while the `rr/gg/bb` represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The `` HTML tag can be used to define the height of the font. Any of the `rrggbb`, mode or

blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<gra FFFFFFFF;1;1>gradient-center</gra>" generates the following picture:

gradient-center

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

outlined

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

shadow

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

outline anti-aliasing

property Item.ToolTipTitle as String

Specifies the title of the item's tooltip.

Type	Description
String	A String expression that specifies the title of the item's tooltip.

The ToolTipTitle property specifies the title for the item's tooltip. The [ToolTip](#) property assigns a HTML tooltip to an item, that's displayed only when the cursor hovers the item. The [ToolTipDelay](#) property specifies the time until the tooltip is shown. Use the [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipWidth](#) property to specify the width of the tooltip window. The [ToolTipFont](#) property specifies 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.

How can I assign a tooltip to an item?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items.Add("ToolTip")
    .ToolTipTitle = "Title"
    .ToolTip = "This is a bit of text that should be shown when cursor hovers the item"
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items.Add("ToolTip")
    .ToolTipTitle = "Title"
    .ToolTip = "This is a bit of text that should be shown when cursor hovers the item"
  End With
  .Refresh
End With
```

VB.NET

```
With Exribbon1
    With .Items.Add("ToolTip")
        .TooltipTitle = "Title"
        .Tooltip = "This is a bit of text that should be shown when cursor hovers the item"
    End With
    .Refresh()
End With
```

VB.NET for /COM

```
With AxRibbon1
    With .Items.Add("ToolTip")
        .TooltipTitle = "Title"
        .Tooltip = "This is a bit of text that should be shown when cursor hovers the item"
    End With
    .Refresh()
End With
```

C++

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemPtr var_Item = spRibbon1->GetItems()-
>Add(L"ToolTip",vtMissing,vtMissing);
var_Item->PutTooltipTitle(L"Title");
var_Item->PutTooltip(L"This is a bit of text that should be shown when cursor
hovers the item");
spRibbon1->Refresh();
```


C++ Builder

```
Exribbonlib_tlb::IItemPtr var_Item = Ribbon1->Items-  
>Add(L"ToolTip",TNoParam(),TNoParam());  
    var_Item->TooltipTitle = L"Title";  
    var_Item->Tooltip = L"This is a bit of text that should be shown when cursor hovers  
the item";  
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Item var_Item = exribbon1.Items.Add("ToolTip",null,null);  
    var_Item.TooltipTitle = "Title";  
    var_Item.Tooltip = "This is a bit of text that should be shown when cursor hovers  
the item";  
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    var var_Item = Ribbon1.Items.Add("ToolTip",null,null);  
        var_Item.TooltipTitle = "Title";  
        var_Item.Tooltip = "This is a bit of text that should be shown when cursor hovers  
the item";  
    Ribbon1.Refresh();  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Ribbon1
    With .Items.Add("ToolTip")
      .TooltipTitle = "Title"
      .Tooltip = "This is a bit of text that should be shown when cursor hovers the
item"
    End With
    .Refresh
  End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXRIBBONLib.Item var_Item = axRibbon1.Items.Add("ToolTip",null,null);
var_Item.TooltipTitle = "Title";
var_Item.Tooltip = "This is a bit of text that should be shown when cursor hovers
the item";
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
  COM com_Item;
  anytype var_Item;
```

```

;

super();

var_Item = COM::createFromObject(exribbon1.Items()).Add("ToolTip"); com_Item =
var_Item;
    com_Item.ToolTipTitle("Title");
    com_Item.ToolTip("This is a bit of text that should be shown when cursor hovers
the item");
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with Items.Add('ToolTip',Nil,Nil) do
    begin
        TooltipTitle := 'Title';
        Tooltip := 'This is a bit of text that should be shown when cursor hovers the item';
    end;
    Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    with Items.Add('ToolTip',Null,Null) do
    begin
        TooltipTitle := 'Title';
        Tooltip := 'This is a bit of text that should be shown when cursor hovers the item';
    end;
    Refresh();
end

```

VFP

```

with thisform.Ribbon1
  with .Items.Add("ToolTip")
    .TooltipTitle = "Title"
    .Tooltip = "This is a bit of text that should be shown when cursor hovers the item"
  endwhile
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item

oRibbon = form.ActiveX1.nativeObject
var_Item = oRibbon.Items.Add("ToolTip")
  var_Item.TooltipTitle = "Title"
  var_Item.Tooltip = "This is a bit of text that should be shown when cursor hovers
the item"
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Item = oRibbon.Items.Add("ToolTip")
  var_Item.TooltipTitle = "Title"
  var_Item.Tooltip = "This is a bit of text that should be shown when cursor hovers
the item"
oRibbon.Refresh()

```

Visual Objects

```

local var_Item as IItem

var_Item := oDCOCX_Exontrol1:Items:Add("ToolTip",nil,nil)

```

```
var_Item:ToolTipTitle := "Title"  
var_Item:ToolTip := "This is a bit of text that should be shown when cursor hovers  
the item"  
oDCOCX_Exontrol1:Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Item  
  
oRibbon = ole_1.Object  
var_Item = oRibbon.Items.Add("ToolTip")  
    var_Item.ToolTipTitle = "Title"  
    var_Item.ToolTip = "This is a bit of text that should be shown when cursor hovers  
the item"  
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate  
    Forward Send OnCreate  
    Variant voltems  
    Get ComItems to voltems  
    Handle holtems  
    Get Create (RefClass(cComItems)) to holtems  
    Set pvComObject of holtems to voltems  
        Variant voltem  
        Get ComAdd of holtems "ToolTip" Nothing Nothing to voltem  
        Handle holtem  
        Get Create (RefClass(cComItem)) to holtem  
        Set pvComObject of holtem to voltem  
            Set ComToolTipTitle of holtem to "Title"  
            Set ComToolTip of holtem to "This is a bit of text that should be shown when  
cursor hovers the item"  
            Send Destroy to holtem  
        Send Destroy to holtems  
    Send ComRefresh
```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oltem := oRibbon:Items():Add("ToolTip")
    oltem:TooltipTitle := "Title"
    oltem:Tooltip := "This is a bit of text that should be shown when cursor hovers
the item"
    oRibbon:Refresh()

    oForm:Show()
    DO WHILE nEvent != xbeP_Quit
        nEvent := AppEvent( @mp1, @mp2, @oXbp )
        oXbp:handleEvent( nEvent, mp1, mp2 )
    ENDDO
    RETURN

```

property Item.ToString as String

Loads or saves the item using string representation.

Type	Description
String	A String expression that specifies the item to be loaded/saved.

The ToString property of the Items object, builds the control/ribbon using a string, rather than adding item one by one. The control's setup installs the WYSIWYG EditRibbon Tool, that helps you to define the ToString format.

The ToString property looks like follows:

```
[id=10][group=0x03]([id=10][group=0x03][itemspad=4,4,4,4][itemsbghot=0x1F000000](Annoyed1[id=20],Bunny2[id=30],[id=50][editttype=0x01][editwidth=-100],Cellphone3[id=40]),[id=10][group=0x03][itemspad=4,4,4,4][itemsbghot=0x1F000000](Annoyed1[id=20],Bunny2[id=30],Cellphone3[id=40]))
```

Each item is followed by its options, and its sub-items between () parentheses. The item's option includes the icons, pictures, edit attributes and so on.

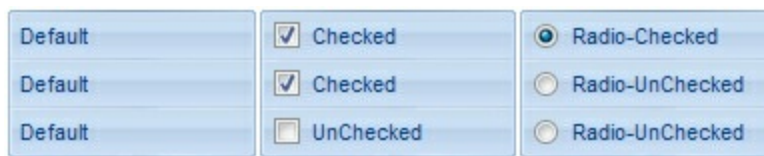
The ToString syntax in BNF notation:

```
<ToString> ::= <ITEMS>
<ITEMS> ::= <ITEM>["("<ITEMS>")"][","<ITEMS>]
<ITEM> ::= <CAPTION>[<OPTIONS>]
<OPTIONS> ::= "["<OPTION>"]"["["<OPTIONS>"]"]
<OPTION> ::= <PROPERTY>["="<VALUE>]
<PROPERTY> ::= "img" | "himg" | "sep" | "id" | "typ" | "group" | "chk" | "button" | "align" |
"spchk" | "show" | "rad" | "dis" | "showdis" | "bld" | "itl" | "stk" | "und" | "bg" | "fg" | "editttype"
| "edit" | "mask" | "border" | "editwidth" | "captionwidth" | "height" | "grp" | "tfi" | "ttp" | "min" | |
|max" | "tick" | "freq" | "ticklabel" | "small" | "large" | "spin" | "ettp" | "float" | "close" | "local" |
| "popupapp" | "itemspad" | "itemsbg" | "itemsbghot" | "itemsbgext" | "visible" | "tab" | "pad" |
| "bghot" | "bgssel" | "bgsselhot" | "arrow" | "popupalign" | "popupoffset" | "popupat" | "hid"
```

where the <CAPTION> is the HTML caption to be shown on the context menu item. The <VALUE> indicates the value of giving property.

- id=<VALUE>, where <VALUE> is an integer expression, that indicates the identifier of the item.
- bg=<VALUE>, specifies the item's background color, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value,

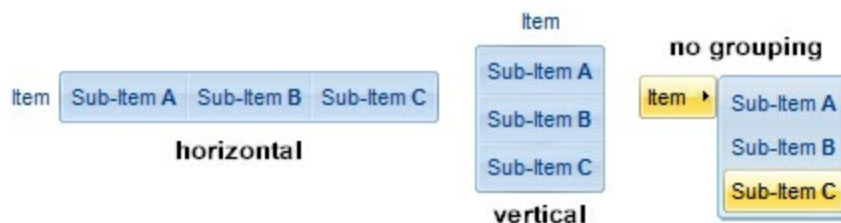
- and the BB is the blue value), or an integer expression to that refers an EBN object.
- **bghot=<VALUE>**, specifies the item's background color, while the cursor hovers the item, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or an integer expression to that refers an EBN object.
- **bgsel=<VALUE>**, specifies the item's background color, while the item is checked/selected, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or an integer expression to that refers an EBN object.
- **bgselhot=<VALUE>**, specifies the item's background color, while the item is checked/selected and the cursor hovers it, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or an integer expression to that refers an EBN object.
- **fg=<VALUE>**, specifies the item's foreground color, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or a integer expression.
- **sep**, specifies an separator item
- **dis**, specifies a disabled item
- **hid**, specifies a hidden item
- **showdis=<VALUE>**, where <VALUE> could be **0** for regular or **not zero** to specify whether the item shows as disabled, but it is still enabled
- **bld**, specifies that the item appears in bold
- **itl**, specifies that the item appears in italics
- **stk**, specifies that the item appears as strikeouts
- **und**, specifies that the item is underlined
- **align=<VALUE>**, where <VALUE> could be one of the following:
 - **0** (left), to align the item's caption to the left
 - **1** (center), to center the item's caption
 - **2** (right), to align the item's caption to the right
- **captionwidth=<VALUE>**, specifies the width to show the HTML caption of the item. where <VALUE> could be a integer expression. A negative value indicates that no limitation is applied to the item's caption, so no truncate caption is shown
- **height=<VALUE>**, specifies the height to show the item, where <VALUE> could be a positive integer expression
- **pad=<VALUE>**, specifies the padding (space between the menu border and the item content) to display the item. The <VALUE> is a list of coordinates such as left,top,right,bottom
- **img=<VALUE>**, where <VALUE> is an integer expression, that indicates the index of the icon being displayed for the item.
- **himg=<VALUE>**, where <VALUE> indicates the key of the picture to be displayed for the item.



- `typ=<VALUE>`, where `<VALUE>` could be one of the following:
 - **0** for default/regular items (no check/radio button is associated with the item),
 - **1** for items that display a check/box (`chk`),
 - **2** to display radio buttons (`rad`)
- `chk[=<VALUE>]`, where `<VALUE>` could be **0** for unchecked, or **not zero** for checked. The `chk` option makes the item to display a check box. If the `<VALUE>` is missing the item still displays an un-checked check box.
- `rad=<VALUE>`, where `<VALUE>` could be **0** for unchecked radio button or **not zero** to for checked radio button. Use the `grp` option to define the group of radio where this button should be associated, If no group of radio buttons is required, the `grp` could be ignored.
- `grp=<VALUE>`, defines the radio group. It should be used when you define more groups of radio buttons. A group of radio buttons means that only one item could be checked at one time. The `rad` option specifies that the item displays a radio button. Use the `grp` option to define the group of radio where this button should be associated, If no group of radio buttons is required, the `grp` could be ignored. The `<VALUE>` could be any integer expression.



- `show=<VALUE>`, where `<VALUE>` could be **0** for regular or **not zero** to specify whether the checked item shows as selected
- `spchk=<VALUE>`, where `<VALUE>` could be **0** for regular or **not zero** to specify whether the item's sub menu is shown only if the item is checked.

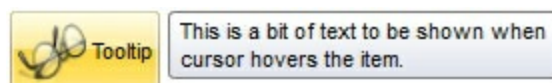


- `group=<VALUE>`, where `<VALUE>` could be a bit-or combination (+) of the following values:
 - **0** (`exNoGroupPopup`), No grouping is performed on the sub-menu, so the sub-items are shown to a float popup,
 - **1** (`exGroupPopup`), Groups and displays the sub-menu items on the current item, arranged from left to right/horizontally

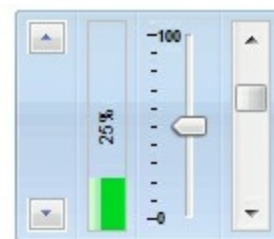
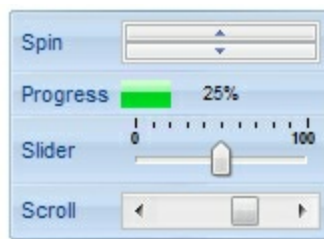
- **2** (exNoGroupPopupFrame), Prevents showing the frame around each grouping item.
- **4** (exGroupPopupCenter), Shows the grouping popup aligned to the center of the current item.
- **8** (exGroupPopupRight), Shows the grouping popup aligned to the right of the current item.
- **16** (exGroupPopupEqualWidth), Shows the items that make the group of the same width
- **32** (exGroupPopupEqualHeight), Shows the items that make the group of the same height
- **64** (exGroupPopupFrameSolidBox), Shows a solid frame around the grouped items
- **128** (exGroupPopupFrameThickBox), Shows a solid thick-frame around the grouped items
- **256** (exGroupPopupVertical), Groups and displays the sub-menu items on the current item, arranged from top to bottom/vertically



- button=<VALUE>, where <VALUE> could be a bit-or combination (+) of the following values.
 - **0** (exShowAsButtonNone), No button is shown,
 - **1** (exShowAsButton), Shows the item as a button
 - **2** (exShowAsButtonAutoSize), Fits the button to cover the item's caption instead showing on the entire item
 - **17** (exShowAsSelectButton), Shows the item as a select button, which is composed by two-fields, one indicates the default button, while the second field specifies the drop down button that displays the items in the current's sub-menu collection. The drop down button is shown to the right-side of the default button. The item must have a submenu, else no drop down is displayed.
 - **273** (exShowAsSelectButtonBottom), Shows the item as a select button, which is composed by two-fields, one indicates the default button, while the second field specifies the drop down button that displays the items in the current's sub-menu collection. The drop down button is shown to the bottom-side of the default button. The item must have a submenu, else no drop down is displayed.



- ttp=<VALUE>, defines the item's tooltip. The <VALUE> could be any HTML string expression. The item's tooltip is shown when the user hovers the item.



- `edittype=<VALUE>`, associates an edit field to the item, where `<VALUE>` could be a combination of one or more of the following values:
 - **0** (`exlItemDisableEdit`), No editor is assigned to the current item.
 - **1** (`exlItemEditText`), A text-box editor is assigned to the current item.
 - **2** (`exlItemEditMask`), A masked text-box editor is assigned to the current item.
 - **3** (`exlItemEditSlider`), A slider editor is assigned to the current item. This can be combined with 1024.
 - **4** (`exlItemEditProgress`), A progress editor is assigned to the current item. This can be combined with 1024.
 - **5** (`exlItemEditScrollBar`), A scrollbar editor is assigned to the current item. This can be combined with 1024.
 - **6** (`exlItemEditColor`), A color editor is assigned to the current item.
 - **7** (`exlItemEditFont`), A font editor is assigned to the current item.
 - **256** (`exlItemEditReadOnly`), specifies that the item's editor is shown as disabled. This value could be combined with one of the values from 0 to 7 or 512
 - **512** (`exlItemEditSpin`), A spin editor is assigned to the current item. This value could be combined with one of the values from 0 to 7 or 256
 - **1024** (`exlItemEditVertical`), The editor is shown vertically rather than horizontally. This value has effect for `exlItemEditSlider`, `exlItemEditProgress` or `exlItemEditScrollBar`
- `edit=<VALUE>`, specifies the caption to be shown in the item's edit field, where `<VALUE>` could be any string
- `mask=<VALUE>`, specifies the mask to be applied on a masked editor. This option is valid for `exlItemEditMask` edit. Use the float option to allow masking floating point numbers. See [Masking](#) for more information about `<VALUE>` of the mask option. See [Masking Float](#) for more information about `<VALUE>` if the float option is used.
- `float=<VALUE>`, Specifies whether the mask field masks a floating point number. This option is valid for `exlItemEditMask` edit. See [Masking Float](#) for more information about `<VALUE>` of mask option, if the float option is used. The `<VALUE>` could be **0** for standard masking field or **not zero** to specify that the field is masking a floating point.
- `border=<VALUE>`, specifies the border to be shown on the item's edit field, where `<VALUE>` could be one of the following:
 - **0** (`exEditBorderNone`), No border is shown.
 - **-1** (`exEditBorderInset`), shows an inset border
 - **1** (`exEditBorderSingle`), shows a frame border
- `editwidth=<VALUE>`, specifies the width to show the edit field inside the item, where `<VALUE>` could be a integer expression. A negative value indicates that the field goes

to the end of the item

- min=<VALUE>, defines the minimum value of the edit field. The <VALUE> could be any integer expression, and specifies the minimum value for any slider, progress, scroll, spin, or range editor.
- max=<VALUE>, defines the maximum value of the edit field. The <VALUE> could be any integer expression, and specifies the maximum value for any slider, progress, scroll, spin, or range editor.
- tick=<VALUE>, defines where the ticks of the slider edit appear. This option is valid for exltemEditSlider edit. The <VALUE> could be one of the following values:
 - **0** (exBottomRight), The ticks are displayed on the bottom/right side.
 - **1** (exTopLeft), The ticks are displayed on the top/left side.
 - **2** (exBoth), The ticks are displayed on the both side.
 - **3** (exNoTicks), No ticks are displayed.
- freq=<VALUE>, indicates the ratio of ticks on the slider edit. This option is valid for exltemEditSlider edit. The <VALUE> could be a positive integer expression.
- ticklabel=<VALUE>, indicates the HTML label to be displayed on slider's ticks. This option is valid for exltemEditSlider edit. See [Tick Label Expression](#) for more information about <VALUE> of the ticklabel option.
- small=<VALUE>, indicates the amount by which the edit's position changes when the user presses the arrow key (left, right, or button). This option is valid for exltemEditSlider, exltemEditScrollBar edit. The <VALUE> could be a positive integer expression.
- large=<VALUE>, indicates the amount by which the edit's position changes when the user presses the CTRL + arrow key (CTRL + left, CTRL + right). This option is valid for exltemEditSlider, exltemEditScrollBar edit. The <VALUE> could be a positive integer expression.
- spin=<VALUE>, specifies the step to advance when user clicks the editor's spin.. This option is valid for exltemEditSpin edit. The <VALUE> could be a positive integer expression.
- ettp=<VALUE>, specifies the HTML tooltip to be shown when the item's value is changed. This option is valid for exltemEditSlider/exltemEditScrollBar edit. The <VALUE> could be any string expression, including built-in HTML tags
- arrow=<VALUE>. The <VALUE> could be **0** for hiding the arrow or **not zero** to show the arrow. Indicates whether an item that has a sub-menu shows or hides its popup arrow. If the <VALUE> is missing, the item shows no arrow.
- local=<VALUE>. The <VALUE> could be **0** for standard popup or **not zero** to specify that the field is a local popup. Specifies whether the item's popup is shown as local. Clicking any item inside a local popup makes the popup itself to close including all its descendent sub-menus, without closing any ascendant sub-menus.
- close=<VALUE>, Specifies the way the hosting menu is closed when the user clicks the item. If the close flag is missing, the <VALUE> is 3 (exCloseOnNonClickable), by default. The <VALUE> could be one of the following values:

- **0** (exCloseOnClick), The popup menu is closing when the user clicks the item.
- **1** (exCloseOnDbClick), The popup menu is closing when the user double clicks the item.
- **2** (exCloseOnClickOutside), The popup menu is closing when the user clicks outside of the menu.
- **3** (exCloseOnNonClickable), The popup menu is closing when the user clicks a non-clickable item (regular items). The non-clickable items is any item that's not a separator, popup, disabled or check or radio items, clicking a check-box item will makes the check box to change its state instead closing the context menu.
- popupapp=<VALUE> indicates the visual appearance of the item's submenu when the popup is shown. The <VALUE> could be a predefine value like shown bellow, or an integer expression that refers an EBN object.
 - **0** (NoBorder)
 - **1** (FlatBorder)
 - **2** (SunkenBorder)
 - **3** (RaisedBorder)
 - **4** (EtchedBorder)
 - **5** (BumpBorder)
 - **6** (ShadowBorder)
 - **7** (InsetBorder)
 - **8** (SingleBorder)
- itemsbg=<VALUE>, specifies the items background color, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or an integer expression to that refers an EBN object.
- itemsbgshot=<VALUE>, specifies the items background color, while the cursor hovers the items, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or an integer expression to that refers an EBN object.
- popupalign=<VALUE>, Indicates how the item's sub-menu is aligned relative to the parent item. The popupalign has no effect for an item that displays a select- button. The <VALUE> could be a combination of one or more of the following values:
 - **0** (exShowPopupAlignNone), The popup menu is shown on top of the item, aligned to the left (no down and right, so up and left)
 - **1** (exShowPopupAlignDown), The popup menu is shown down. If missing, the popup menu is shown up.
 - **2** (exShowPopupAlignRight), The popup menu is shown aligned to the right, else if missing, the popup menu is shown aligned to the left.
- popupat=<VALUE>, specifies the identifier of the item where the current item's submenu/popup is displayed. The <VALUE> could be any integer expression. If there is no identifier with giving value, the option has no effect.
- popupoffset=<VALUE>, specifies the offset (horizontal,vertical) to display the item's

submenu/popup relative to its default position.

- `itemspad=<VALUE>`, specifies the padding (space between the menu border and the item content) to display the items. The `<VALUE>` is a list of coordinates such as `left,top,right,bottom`
- `visible=<VALUE>`, specifies the maximum number of visible items at one time, where the `<VALUE>` could be any integer expression.
- `tab=<VALUE>`, specifies the identifier of the item/tab where the current group-popup is shown instead. The `<VALUE>` could be any integer expression. If there is no identifier with giving value, the option has no effect.
- `itemsbgext=<VALUE>`, indicates additional colors, text, images that can be displayed on the items background using the [EBN String Format](#). The `<VALUE>` should be in [EBN String Format](#). For instance, `[itemsbgext=bottom[2],bottom[16,text=`</fgcolor><fgcolor 6D6AAA>Views</fgcolor><fgcolor A0A0A0>`,align=0x21]]`, shows the Views aligned to the bottom, with a different foreground color.

Masking, (mask option)

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

!(999) 000 0000;1;;;select=1,empty,overtypewarning=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 *overtypewarning* 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;overtypewarning=<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 use 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)
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 overtyping option, it indicates that the user can toggle the insert/over-type mode using the Insert key. For instance, the "###:###;0;inserttype,overtyping", 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.
- **overtyping**, 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;overtyping,inserttype", indicates that the field enter in over-type mode, and insert-type mode is allowed. The "###:###;0;overtyping", 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.
- **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,overtyp"`, 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,overtyp,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

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.

Masking-Float, (mask, float option)

The [mask=<VALUE>] property may indicate the followings, if the [float=-1] is present

- **negative number**: if the first character in the mask is - (minus) the control supports negative numbers. Pressing the - key will toggle the sign of the number. The + sign is never displayed.
- **decimal symbol**: the last character that's different than # (digit), or 0 (zero) indicates the decimal symbol. If it is not present the control mask a floating point number without decimals.
- **thousand symbol**: the thousand symbol is the last character that's not a # (digit), 0 (zero) or it is not the decimal symbol as explained earlier, if present.
- the maximum **number of decimals** in the number (the # or 0 character after the decimal symbol)
- the maximum number of digits in the integer part (the number of # or 0 character before decimal symbol)
- the **0** character indicates a **leading-zero**. The count of 0 (zero) characters before decimal character indicates the leading-zero for integer part of the control, while the count of 0 (zero) characters after the decimal separator indicates the leading-zero for decimal part of the control. For instance, the Mask on "-###,###,##0.00", while the control's Text property is 1, the control displays 1.00, if 1.1 if displays 1.10, and if empty, the 0.00 is displayed.

If the <VALUE> property is empty, the control takes the settings for the regional options like: Decimal Symbol , No. of digits after decimal, Digit grouping symbol.

Here are few samples:

The <VALUE>"-###.###.##0,00" filter floating point numbers a number for German settings ("," is the decimal sign, "." is the thousands separator). This format displays leading-zeros.

The <VALUE>"-###.###.###,##" filter floating point numbers a number for German settings ("," is the decimal sign, "." is the thousands separator)

The <VALUE>"-###,###,###.##" filter floating point numbers a number for English settings ("." is the decimal sign, "," is the thousands separator)

The <VALUE>"####" indicates a max-4 digit number (positive) without a decimal symbol and without digit grouping

The <VALUE>"-##.##" filters a floating point number from the -99.9 to 99.9 ("." is the decimal sign, no thousands separator)

The <VALUE>"#,###.##" filters a floating point number from the 0 to 9,999.99 with digit grouping ("." is the decimal sign, "," is the thousands separator).

Tick Label Expression, (ticklabel option)



For instance:

- "value", shows the values for each tick.
- "(value=current ? '<fgcolor=FF0000>' : ") + value", shows the current slider's position with a different color and font.
- "value = current ? value : """, shows the value for the current tick only.
- "(value = current ? '' : ") + (value array 'ab bc cd de ef fg gh hi ij jk kl' split ' ')" displays different captions for slider's values.

The The <VALUE> of [ticklabel] option is a formatted expression which result may include the [HTML](#) tags.

The The <VALUE> of [ticklabel] option indicates a formatting expression that may use the following predefined keywords:

- **value** gets the slider's position to be displayed
- **current** gets the current slider's value.
- **vmin** gets the slider's minimum value.
- **vmax** gets the slider's maximum value.
- **smin** gets the slider's selection minimum value.
- **smax** gets the slider's selection maximum value.

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 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 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 than *iif* (immediate if operator) alternative.

- **case()** (*case operator*) returns and executes one of n expressions, depending on the evaluation of the expression (*IIF* - immediate IF operator is a binary case() operator). The syntax for *case()* operator is:

"expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)"

If the default part is missing, the *case()* operator returns the value of the expression if it is not found in the collection of cases (c1, c2, ...). For instance, if the value of

expression is not any of c1, c2, the default_expression is executed and returned. If the value of the expression is c1, then the case() operator executes and returns the expression1. The default, c1, c2, c3, ... must be constant elements as numbers, dates or strings. For instance, the "date(shortdate(value)) case (default:0 ; #1/1/2002#:1 ; #2/1/2002#:1; #4/1/2002#:1; #5/1/2002#:1)" indicates that only #1/1/2002#, #2/1/2002#, #4/1/2002# and #5/1/2002# dates returns 1, since the others returns 0. For instance the following sample specifies the hour being non-working for specified dates: "date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)" statement indicates the working hours for dates as follows:

- - #4/1/2009#, from hours 06:00 AM to 12:00 PM
 - #4/5/2009#, from hours 07:00 AM to 10:00 AM and hours 03:00PM, 04:00PM, 06:00PM and 10:00PM
 - #5/1/2009#, from hours 12:00 AM to 08:00 AM

The in, switch and case() use binary search to look for elements so they are faster 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. For instance type(%0) = 8 specifies the cells that contains string values.

Here's few predefined types:

- 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
- **str** (unary operator) converts the expression to a string
- **dbl** (unary operator) converts the expression to a number
- **date** (unary operator) converts the expression to a date, based on your regional settings
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS.

Other known operators for numbers are:

- **int** (unary operator) retrieves the integer part of the number
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2
- 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

Other known operators for strings are:

- **len** (unary operator) retrieves the number of characters in the string
- **lower** (unary operator) returns a string expression in lowercase letters
- **upper** (unary operator) returns a string expression in uppercase letters
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names
- **ltrim** (unary operator) removes spaces on the left side of a string
- **rtrim** (unary operator) removes spaces on the right side of a string
- **trim** (unary operator) removes spaces on both sides of a string
- **startswith** (binary operator) specifies whether a string starts with specified string
- **endwith** (binary operator) specifies whether a string ends with specified string
- **contains** (binary operator) specifies whether a string contains another specified string
- **left** (binary operator) retrieves the left part of the string
- **right** (binary operator) retrieves the right part of the string
- a **mid** b (binary operator) retrieves the middle part of the string a starting from b (1 means first position, and so on)
- a **count** b (binary operator) retrieves the number of occurrences of the b in a
- a **replace** b **with** c (double binary operator) replaces in a the b with c, and gets the result.
- a **split** b, 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

Other known operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel.
- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance the timeF(1:23 PM) returns "13:23:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel.
- **shortdateF** (unary operator) formats a date as a date string using the "MM/DD/YYYY" format. For instance the shortdateF(December 31, 1971 11:00 AM) returns "12/31/1971".
- **dateF** (unary operator) converts the date expression to a string expression in "MM/DD/YYYY HH:MM:SS" format.
- **longdate** (unary operator) formats a date as a date string using the long date format, as specified in the control's panel.
- **year** (unary operator) retrieves the year of the date (100,...,9999)
- **month** (unary operator) retrieves the month of the date (1, 2,...,12)
- **day** (unary operator) retrieves the day of the date (1, 2,...,31)
- **yearday** (unary operator) retrieves the number of the day in the year, or the days since January 1st (0, 1,...,365)
- **weekday** (unary operator) retrieves the number of days since Sunday (0 - Sunday, 1 - Monday,..., 6 - Saturday)
- **hour** (unary operator) retrieves the hour of the date (0, 1, ..., 23)
- **min** (unary operator) retrieves the minute of the date (0, 1, ..., 59)
- **sec** (unary operator) retrieves the second of the date (0, 1, ..., 59)

The The <VALUE> of [ticklabel] option can display labels using the following built-in HTML tags:

- **** displays the text in **bold**.
- **<i></i>** displays the text in *italics*.
- **<u></u>** underlines the text.
- **<s></s>** Strike-through text
- **** displays portions of text with a different font and/or different size. For instance, the bit draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, bit displays the bit text using the current font, but with a different size.
- **<fgcolor=RRGGBB></fgcolor>** displays text with a specified **foreground** color. The RR, GG or BB should be hexa values and indicates red, green and blue values.
- **<bgcolor=RRGGBB></bgcolor>** displays text with a specified **background** color. The RR, GG or BB should be hexa values and indicates red, green and blue values.
- **
** a forced line-break
- **<solidline>** The next line shows a solid-line on top/bottom side. If has no effect for a

single line caption.

- **<dotline>** The next line shows a dot-line on top/bottom side. If has no effect for a single line caption.
- **<upline>** The next line shows a solid/dot-line on top side. If has no effect for a single line caption.
- **<r>** Right aligns the text
- **<c>** Centers the text
- **number[:width]** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **key[:width]** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&**; (&), **<**; (<), **>**; (>), **&qout;** (") and **&#number** (the character with specified code), For instance, the **€** displays the EUR character, in UNICODE configuration. The **&** ampersand is only recognized as markup when it is followed by a known letter or a # character and a digit. For instance if you want to display **bold** in HTML caption you can use **bold**;

EBN String Format, (itemsbgext option)

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

```
<EBN> ::= <elements> | <root> "(" [<elements>] ")"
<elements> ::= <element> [ "," <elements> ]
<root> ::= "root" [ <attributes> ] | [ <attributes> ]
<element> ::= <anchor> [ <attributes> ] [ "(" [<elements>] ")" ]
<anchor> ::= "none" | "left" | "right" | "client" | "top" | "bottom"
<attributes> ::= "[" [<client> ","] <attribute> [ "," <attributes> ] "]"
<client> ::= <expression> | <expression> "," <expression> "," <expression> ","
<expression>
<expression> ::= <number> | <number> "%"
<attribute> ::= <backcolor> | <text> | <wordwrap> | <align> | <pattern> |
<patterncolor> | <frame> | <framethick> | <data> | <others>
```

```

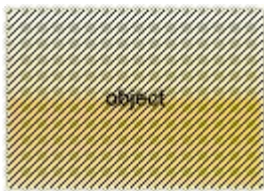
<equal> ::= "="
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<decimal> ::= <digit> <decimal>
<hexadigit> ::= <digit> | "A" | "B" | "C" | "D" | "E" | "F"
<hexa> ::= <hexadigit> <hexa>
<number> ::= <decimal> | "0x" <hexa>
<color> ::= <rgbcolor> | number
<rgbcolor> ::= "RGB" "(" <number> "," <number> "," <number> ")"
<string> ::= "\"" <characters> "\"" | "'" <characters> "'" | "<characters> "
<characters> ::= <char> | <characters>
<char> ::= <any_character_excepts_null>
<bgcolor> ::= "back" <equal> <color>
<text> ::= "text" <equal> <string>
<align> ::= "align" <equal> <number>
<pattern> ::= "pattern" <equal> <number>
<patterncolor> ::= "patterncolor" <equal> <color>
<frame> ::= "frame" <equal> <color>
<data> ::= "data" <equal> <number> | <string>
<framethick> ::= "framethick"
<wordwrap> ::= "wordwrap"

```

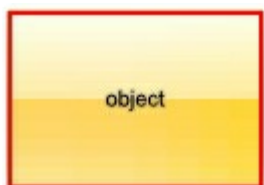
Others like: pic, stretch, hstretch, vstretch, transparent, from, to are reserved for future use only.

Here's a few easy samples:

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

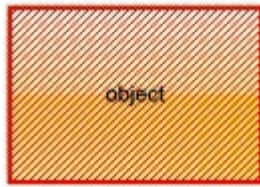


- "[frame=RGB(255,0,0),framethick]", draws a red thick-border around the object.

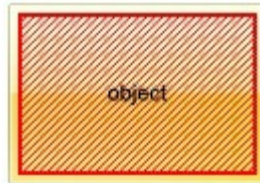


- "[frame=RGB(255,0,0),framethick,pattern=6,patterncolor=RGB(255,0,0)]", draws a

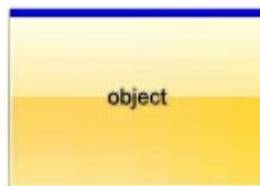
red thick-border around the object, with a patter inside.



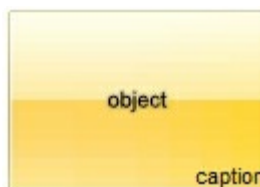
- "[[patterncolor=RGB(255,0,0)]
(none[(4,4,100%-8,100%-8),pattern=0x006,patterncolor=RGB(255,0,0),frame=RGB(255,0,0)])]" draws a red thick-border around the object, with a patter inside, with a 4-pixels wide padding:



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



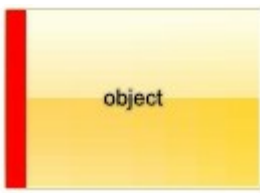
- "[text=`caption`,align=0x22)]", shows the caption string aligned to the bottom-right side of the object's background.



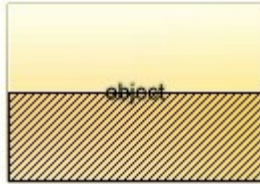
- "[text=`flag`,align=0x11)]" shows the flag picture and the sweden string aligned to the bottom side of the object.



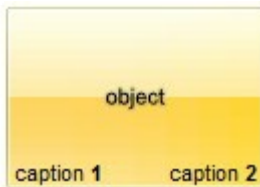
- "left[10,back=RGB(255,0,0)]", draws a red line on the left side of the object's background, of 10-pixels wide.



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



- "root[text=`caption 2`,align=0x22](client[text=`caption 1`,align=0x20])", shows the caption **1** aligned to the bottom-left side, and the caption **2** to the bottom-right side



property Item.Underline as Boolean

Specifies whether the item's caption appears as underlined.

Type	Description
Boolean	A Boolean expression that specifies whether the item's caption is underlined.

By default, the Underline property is False. Use the Underline property to show underlined the item's caption. The [Caption](#) property indicates the HTML caption to be shown on the item. The <u> HTML tag can be used on the item's Caption property to specify different parts of the caption as underlined.

How can I underline the item?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    .Add("Item").Underline = True
    .Add "<u>Item</u>"
    .Add("").ToString = "Item[und]"
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    .Add("Item").Underline = True
    .Add "<u>Item</u>"
    .Add("").ToString = "Item[und]"
  End With
  .Refresh
End With
```

VB.NET

```
With Exribbon1
  With .Items
```

```

.Add("Item").Underline = True
.Add("<u>Item</u>")
.Add("").ToString = "Item[und]"
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    With .Items
        .Add("Item").Underline = True
        .Add("<u>Item</u>")
        .Add("").ToString = "Item[und]"
    End With
    .Refresh()
End With

```

C++

```

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
var_Items->Add(L"Item",vtMissing,vtMissing)->PutUnderline(VARIANT_TRUE);
var_Items->Add(L"<u>Item</u>",vtMissing,vtMissing);
var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"Item[und]");
spRibbon1->Refresh();

```

C++ Builder


```

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
var_Items->Add(L"Item",TNoParam(),TNoParam())->Underline = true;
var_Items->Add(L"<u>Item</u>",TNoParam(),TNoParam());
var_Items->Add(L"",TNoParam(),TNoParam())->ToString = L"Item[und]";
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
var_Items.Add("Item",null,null).Underline = true;
var_Items.Add("<u>Item</u>",null,null);
var_Items.Add("",null,null).ToString = "Item[und]";
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var_Items.Add("Item",null,null).Underline = true;
    var_Items.Add("<u>Item</u>",null,null);
    var_Items.Add("",null,null).ToString = "Item[und]";
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>

```

```

<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Ribbon1
    With .Items
      .Add("Item").Underline = True
      .Add "<u>Item</u>"
      .Add("").ToString = "Item[und]"
    End With
    .Refresh
  End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.Add("Item",null,null).Underline = true;
var_Items.Add("<u>Item</u>",null,null);
var_Items.Add("",null,null).ToString = "Item[und]";
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
  COM com_Item,com_Items;
  anytype var_Item,var_Items;
  ;

  super();

  var_Items = exribbon1.Items(); com_Items = var_Items;

```

```

    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
    com_Item.Underline(true);
    com_Items.Add("<u>Item</u>");
    var_Item = COM::createFromObject(com_Items.Add("")); com_Item = var_Item;
    com_Item.ToString("Item[und]");
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
  with Items do
  begin
    Add('Item',Nil,Nil).Underline := True;
    Add('<u>Item</u>',Nil,Nil);
    Add('',Nil,Nil).ToString := 'Item[und]';
  end;
  Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  with Items do
  begin
    Add('Item',Null,Null).Underline := True;
    Add('<u>Item</u>',Null,Null);
    Add('',Null,Null).ToString := 'Item[und]';
  end;
  Refresh();
end

```

VFP

```

with thisform.Ribbon1

```

```

with .Items
    .Add("Item").Underline = .T.
    .Add("<u>Item</u>")
    .Add("").ToString = "Item[und]"
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Items

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
// var_Items.Add("Item").Underline = true
var_Item = var_Items.Add("Item")
with (oRibbon)
    TemplateDef = [Dim var_Item]
    TemplateDef = var_Item
    Template = [var_Item.Underline = true]
endwith
var_Items.Add("<u>Item</u>")
// var_Items.Add("").ToString = "Item[und]"
var_Item1 = var_Items.Add("")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.ToString = "Item[und]"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P

```

```

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
' var_Items.Add("Item").Underline = .t.
var_Item = var_Items.Add("Item")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.Underline = True"

var_Items.Add("<u>Item</u>")
' var_Items.Add("").ToString = "Item[und]"
var_Item1 = var_Items.Add("")
oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1
oRibbon.Template = "var_Item1.ToString = \"Item[und]\""

oRibbon.Refresh()

```

Visual Objects

```

local var_Items as IItems

var_Items := oDCOCX_Exontrol1.Items
var_Items.Add("Item",nil,nil):Underline := true
var_Items.Add("<u>Item</u>",nil,nil)
var_Items.Add("",nil,nil):ToString := "Item[und]"
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```

OleObject oRibbon,var_Items

oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Items.Add("Item").Underline = true
var_Items.Add("<u>Item</u>")

```

```
var_Items.Add("").ToString = "Item[und]"  
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate  
    Forward Send OnCreate  
    Variant voltems  
    Get ComItems to voltems  
    Handle holtems  
    Get Create (RefClass(cComItems)) to holtems  
    Set pvComObject of holtems to voltems  
        Variant voltem  
        Get ComAdd of holtems "Item" Nothing Nothing to voltem  
        Handle holtem  
        Get Create (RefClass(cComItem)) to holtem  
        Set pvComObject of holtem to voltem  
            Set ComUnderline of holtem to True  
        Send Destroy to holtem  
        Get ComAdd of holtems "<u>Item</u>" Nothing Nothing to Nothing  
        Variant voltem1  
        Get ComAdd of holtems "" Nothing Nothing to voltem1  
        Handle holtem1  
        Get Create (RefClass(cComItem)) to holtem1  
        Set pvComObject of holtem1 to voltem1  
            Set ComToString of holtem1 to "Item[und]"  
        Send Destroy to holtem1  
    Send Destroy to holtems  
    Send ComRefresh  
End_Procedure
```

XBase++

```
#include "AppEvent.ch"  
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

LOCAL oForm

LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

LOCAL oltems

LOCAL oRibbon

oForm := XbpDialog():new(AppDesktop())

oForm:drawingArea:clipChildren := .T.

oForm:create(,, {100,100}, {640,480},,, .F.)

oForm:close := {|| PostAppEvent(xbeP_Quit)}

oRibbon := XbpActiveXControl():new(oForm:drawingArea)

oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-CFBE431702E2}*/

oRibbon:create(,, {10,60},{610,370})

oltems := oRibbon:Items()

oltems:Add("Item"):Underline := .T.

oltems:Add("<u>Item</u>")

oltems:Add(""):ToString := "Item[und]"

oRibbon:Refresh()

oForm:Show()

DO WHILE nEvent != xbeP_Quit

nEvent := AppEvent(@mp1, @mp2, @oXbp)

oXbp:handleEvent(nEvent, mp1, mp2)

ENDDO

RETURN

method Item.Update ()

Updates the item.

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

Use the Update method to update a particular item. The Update method does not re-computes the required size of the item, so it just validates the item's client area. The [Update](#) method invalidates the entire control, without resizing the elements inside. The [Refresh](#) method refreshes the entire control, including resizing inside elements.

property Item.UserData as Variant

Associates an extra data to the object.

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

By default, the UserData is empty. Use the UserData property to associate any extra data to the item. Use the [Caption](#) property to specify the item's caption. Use the [Tooltip](#) property to specify the item's tooltip which can be shown when the cursor hovers the item. The [Item](#) property searches recursively the item with giving identifier/caption.

property Item.Visible as Boolean

Specifies whether the item is visible or hidden.

Type	Description
Boolean	A Boolean expression that specifies whether the item is visible or hidden.

By default, the Visible property is True. You can use the Visible property to show or hide the item. Use the [Refresh](#) method to update the control, so the item became hidden. Use the [Enabled](#) property to disable an item. A disabled item shows as grayed, and it is unselectable, so the user can select or highlight it. The [Remove](#) method removes an individual Item object giving its identifier or caption.

How can I hide an item?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    .Add("Item 1").Visible = False
    .Add "Item 2"
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    .Add("Item 1").Visible = False
    .Add "Item 2"
  End With
  .Refresh
End With
```

VB.NET

```
With Exribbon1
  With .Items
    .Add("Item 1").Visible = False
```

```
.Add("Item 2")
End With
.Refresh()
End With
```

VB.NET for /COM

```
With AxRibbon1
    With .Items
        .Add("Item 1").Visible = False
        .Add("Item 2")
    End With
    .Refresh()
End With
```

C++

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
var_Items->Add(L"Item 1",vtMissing,vtMissing)->PutVisible(VARIANT_FALSE);
var_Items->Add(L"Item 2",vtMissing,vtMissing);
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
var_Items->Add(L"Item 1",TNoParam(),TNoParam())->Visible = false;
var_Items->Add(L"Item 2",TNoParam(),TNoParam());
```

```
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
    var_Items.Add("Item 1",null,null).Visible = false;  
    var_Items.Add("Item 2",null,null);  
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload= 'Init()'>  
<OBJECT CLASSID= "clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id= "Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE= "JScript">  
function Init()  
{  
    var var_Items = Ribbon1.Items;  
    var_Items.Add("Item 1",null,null).Visible = false;  
    var_Items.Add("Item 2",null,null);  
    Ribbon1.Refresh();  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload= 'Init()'>  
<OBJECT CLASSID= "clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id= "Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE= "VBScript">  
Function Init()  
    With Ribbon1
```

```

With .Items
    .Add("Item 1").Visible = False
    .Add "Item 2"
End With
.Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.Add("Item 1",null,null).Visible = false;
var_Items.Add("Item 2",null,null);
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = COM::createFromObject(com_Items.Add("Item 1")); com_Item =
var_Item;
    com_Item.Visible(false);
    com_Items.Add("Item 2");
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
  with Items do
  begin
    Add('Item 1',Nil,Nil).Visible := False;
    Add('Item 2',Nil,Nil);
  end;
  Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  with Items do
  begin
    Add('Item 1',Null,Null).Visible := False;
    Add('Item 2',Null,Null);
  end;
  Refresh();
end

```

VFP

```

with thisform.Ribbon1
  with .Items
    .Add("Item 1").Visible = .F.
    .Add("Item 2")
  endwith
  .Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Items

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items

```

```
// var_Items.Add("Item 1").Visible = false
var_Item = var_Items.Add("Item 1")
with (oRibbon)
    TemplateDef = [Dim var_Item]
    TemplateDef = var_Item
    Template = [var_Item.Visible = false]
endwith
var_Items.Add("Item 2")
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
Dim var_Item as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
' var_Items.Add("Item 1").Visible = .f.
var_Item = var_Items.Add("Item 1")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.Visible = False"

var_Items.Add("Item 2")
oRibbon.Refresh()
```

Visual Objects

```
local var_Items as IItems

var_Items := oDCOCX_Exontrol1:Items
var_Items:Add("Item 1",nil,nil):Visible := false
var_Items:Add("Item 2",nil,nil)
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Items  
  
oRibbon = ole_1.Object  
var_Items = oRibbon.Items  
    var_Items.Add("Item 1").Visible = false  
    var_Items.Add("Item 2")  
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate  
    Forward Send OnCreate  
    Variant voltems  
    Get ComItems to voltems  
    Handle holtems  
    Get Create (RefClass(cComItems)) to holtems  
    Set pvComObject of holtems to voltems  
        Variant voltem  
        Get ComAdd of holtems "Item 1" Nothing Nothing to voltem  
        Handle holtem  
        Get Create (RefClass(cComItem)) to holtem  
        Set pvComObject of holtem to voltem  
            Set ComVisible of holtem to False  
        Send Destroy to holtem  
        Get ComAdd of holtems "Item 2" Nothing Nothing to Nothing  
    Send Destroy to holtems  
    Send ComRefresh  
End_Procedure
```

XBase++

```
#include "AppEvent.ch"  
#include "ActiveX.ch"  
  
PROCEDURE Main
```


LOCAL oForm

LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

LOCAL oltems

LOCAL oRibbon

oForm := XbpDialog():new(AppDesktop())

oForm:drawingArea:clipChildren := .T.

oForm:create(,, {100,100}, {640,480},,, .F.)

oForm:close := {|| PostAppEvent(xbeP_Quit)}

oRibbon := XbpActiveXControl():new(oForm:drawingArea)

oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-CFBE431702E2}*/

oRibbon:create(,, {10,60},{610,370})

oltems := oRibbon:Items()

oltems:Add("Item 1"):Visible := .F.

oltems:Add("Item 2")

oRibbon:Refresh()

oForm:Show()

DO WHILE nEvent != xbeP_Quit

nEvent := AppEvent(@mp1, @mp2, @oXbp)

oXbp:handleEvent(nEvent, mp1, mp2)

ENDDO

RETURN

Items object

The Items collection supports the following properties and methods:

Name	Description
Add	Adds an Item object and returns a reference to the newly created object.
BackColor	Specifies the background color of the items.
BackgroundExt	Indicates additional colors, text, images that can be displayed on the items's background using the EBN string format.
Clear	Removes all objects in a collection.
Count	Returns the number of objects in a collection.
HotBackColor	Specifies the hot background color of the items (when the cursor hovers the items).
Item	Returns a specific Item object giving its identifier.
Padding	Specifies the padding (space between the menu border and the item content) to display the items.
PopupAppearance	Retrieves or sets the popup's appearance.
Remove	Removes a specific member from the collection.
SortOrder	Sorts the items in the submenu.
ToString	Loads or saves the Items collection using string representation.
VisibleItemsCount	Specifies the maximum number of visible items at one time.

method Items.Add (Caption as String, [ItemType as Variant], [ID as Variant])

Adds an Item object and returns a reference to the newly created object.

Type	Description
Caption as String	A String expression that specifies the HTML caption to be displayed on the item.
ItemType as Variant	An ItemTypeEnum expression that specifies the type of the item to be added.
ID as Variant	A Long expression that specifies the identifier of the item to be added.

Return	Description
Item	An Item object being created.

The Add method adds a new item to the Items collection. The [ToString](#) property loads or saves the control items from a string, so you can use the ToString method to add items too!. The [Remove](#) method removes a specified item. The [Item](#) property gets the Item object giving its identifier or caption. The [SubMenu](#) property gets a collection of Item objects to be displayed on the sub-menu. This property returns a not-empty value, if the ItemType parameter is SubMenu. The [SubControl](#) property gets access to the [Control](#) object that holds information about the inside ActiveX or Window hosted by the item. This property returns a not-empty value, if the ItemType parameter is SubControl.

The Caption parameter supports the following HTML tags:

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

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

such as: Text with subscript The "Text with <off -6>superscript" displays the text such as: Text with subscript

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

gradient-center

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

outlined

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

shadow

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

outline anti-aliasing

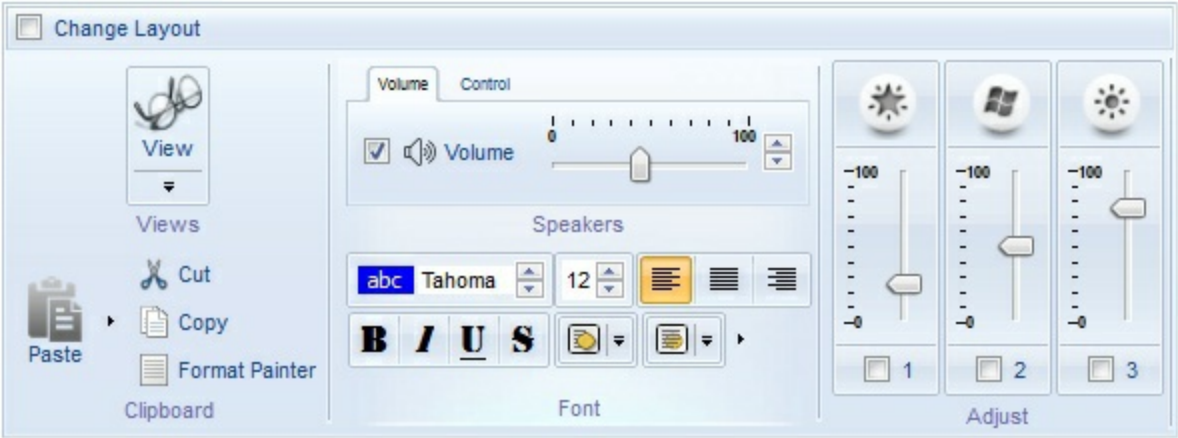
property Items.BackColor as Color

Specifies the background color of the items.

Type	Description
Color	A Color expression that specifies the 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.

The BackColor property specifies the solid color / visual appearance to be shown on the items' background (inside borders). The [BackgroundExt](#) property indicates additional colors, text, images that can be displayed on the items's background using the EBN string format. The [PopupAppearance](#) property specifies the visual appearance of the items (including the margins/borders). The [HotBackColor](#) property specifies the background color for items when cursor hovers it. The [Padding](#) property specifies the padding (space between the menu border and the item content) to display the items.

The following screen shot shows different grouping items (Clipboard, Font, Adjust) with different background appearance.



How can I change the visual appearance for items (EBN)?

VBA (MS Access, Excell...)

With Ribbon1

.VisualAppearance.Add 1,"c:\exontrol\images\normal.ebn"

With .Items

With .Add("",2)

.GroupPopup = 3 ' **GroupPopupEnum.exNoGroupPopupFrame Or**

GroupPopupEnum.exGroupPopup

With .Items

.**BackColor** = &H1000000

.Padding = "4,8,4,8"

.Add "Item 1"

.Add "Item 2"

.Add "Item 3"

End With

End With

.Add("").ToString = "[group=0x03][itemspad=4,8,4,8][itemsbg=0x1000000](Item 1,Item 2,Item 3)"

End With

.Refresh

End With

VB6

With Ribbon1

.**VisualAppearance**.Add 1,"c:\exontrol\images\normal.ebn"

With .Items

With .Add("",2)

.GroupPopup = GroupPopupEnum.exNoGroupPopupFrame Or

GroupPopupEnum.exGroupPopup

With .Items

.**BackColor** = &H1000000

.Padding = "4,8,4,8"

.Add "Item 1"

.Add "Item 2"

.Add "Item 3"

End With

End With

.Add("").ToString = "[group=0x03][itemspad=4,8,4,8][itemsbg=0x1000000](Item 1,Item 2,Item 3)"

End With

.Refresh

End With

VB.NET

With Exribbon1

.VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")

With .Items

With .Add("",2)

.GroupPopup =

exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or

exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup

With .Items

.BackColor32 = &H1000000

.Padding = "4,8,4,8"

.Add("Item 1")

.Add("Item 2")

.Add("Item 3")

End With

End With

.Add("").ToString = "[group=0x03][itemspad=4,8,4,8][itemsbg=0x1000000](Item 1,Item 2,Item 3)"

End With

.Refresh()

End With

VB.NET for /COM

With AxRibbon1

.VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")

With .Items

With .Add("",2)

.GroupPopup = EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or

EXRIBBONLib.GroupPopupEnum.exGroupPopup

With .Items

.BackColor = &H1000000

.Padding = "4,8,4,8"

.Add("Item 1")

.Add("Item 2")

.Add("Item 3")

End With

End With


```

        .Add("").ToString = "[group=0x03][itemspad=4,8,4,8][itemsbg=0x1000000](Item
1,Item 2,Item 3)"
    End With
    .Refresh()
End With

```

C++

```

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
spRibbon1->GetVisualAppearance()->Add(1,"c:\\exontrol\\images\\normal.ebn");
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"",long(2),vtMissing);
    var_Item-
>PutGroupPopup(EXRIBBONLib::GroupPopupEnum(EXRIBBONLib::exNoGroupPopupF
| EXRIBBONLib::exGroupPopup));
    EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();
    var_Items1->PutBackColor(0x1000000);
    var_Items1->PutPadding(L"4,8,4,8");
    var_Items1->Add(L"Item 1",vtMissing,vtMissing);
    var_Items1->Add(L"Item 2",vtMissing,vtMissing);
    var_Items1->Add(L"Item 3",vtMissing,vtMissing);
    var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"[group=0x03]
[itemspad=4,8,4,8][itemsbg=0x1000000](Item 1,Item 2,Item 3)");
spRibbon1->Refresh();

```

C++ Builder

```

Ribbon1->VisualAppearance-

```

```

>Add(1,TVariant("c:\\exontrol\\images\\normal.ebn"));
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    Exribbonlib_tlb::IItemPtr var_Item = var_Items->Add(L"",TVariant(2),TNoParam());
    var_Item->GroupPopup =
Exribbonlib_tlb::GroupPopupEnum::exNoGroupPopupFrame |
Exribbonlib_tlb::GroupPopupEnum::exGroupPopup;
    Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;
    var_Items1->BackColor = 0x1000000;
    var_Items1->Padding = L"4,8,4,8";
    var_Items1->Add(L"Item 1",TNoParam(),TNoParam());
    var_Items1->Add(L"Item 2",TNoParam(),TNoParam());
    var_Items1->Add(L"Item 3",TNoParam(),TNoParam());
    var_Items->Add(L"",TNoParam(),TNoParam())->ToString = L"[group=0x03]
[itemspad=4,8,4,8][itemsbg=0x1000000](Item 1,Item 2,Item 3)";
Ribbon1->Refresh();

```

C#

```

exribbon1.VisualAppearance.Add(1,"c:\\exontrol\\images\\normal.ebn");
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
    exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
    var_Item.GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup;
    exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
    var_Items1.BackColor32 = 0x1000000;
    var_Items1.Padding = "4,8,4,8";
    var_Items1.Add("Item 1",null,null);
    var_Items1.Add("Item 2",null,null);
    var_Items1.Add("Item 3",null,null);
    var_Items.Add("",null,null).ToString = "[group=0x03][itemspad=4,8,4,8]
[itemsgb=0x1000000](Item 1,Item 2,Item 3)";
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Ribbon1.VisualAppearance.Add(1,"c:\\exontrol\\images\\normal.ebn");
    var var_Items = Ribbon1.Items;
    var var_Item = var_Items.Add("",2,null);
    var_Item.GroupPopup = 3;
    var var_Items1 = var_Item.Items;
    var_Items1.BackColor = 16777216;
    var_Items1.Padding = "4,8,4,8";
    var_Items1.Add("Item 1",null,null);
    var_Items1.Add("Item 2",null,null);
    var_Items1.Add("Item 3",null,null);
    var_Items.Add("",null,null).ToString = "[group=0x03][itemspad=4,8,4,8]
[itemsbg=0x1000000](Item 1,Item 2,Item 3)";
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        .VisualAppearance.Add 1,"c:\\exontrol\\images\\normal.ebn"
    With .Items
        With .Add("",2)

```

.GroupPopup = 3 ' **GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup**

With .Items

.**BackColor** = &H1000000

.Padding = "4,8,4,8"

.Add "Item 1"

.Add "Item 2"

.Add "Item 3"

End With

End With

.Add("").ToString = "[group=0x03][itemspad=4,8,4,8][itemsbg=0x1000000]
(Item 1,Item 2,Item 3)"

End With

.Refresh

End With

End Function

</SCRIPT>

</BODY>

C# for /COM

```
axRibbon1.VisualAppearance.Add(1,"c:\\exontrol\\images\\normal.ebn");  
EXRIBBONLib.Items var_Items = axRibbon1.Items;  
EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);  
var_Item.GroupPopup =  
EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |  
EXRIBBONLib.GroupPopupEnum.exGroupPopup;  
EXRIBBONLib.Items var_Items1 = var_Item.Items;  
var_Items1.BackColor = 0x1000000;  
var_Items1.Padding = "4,8,4,8";  
var_Items1.Add("Item 1",null,null);  
var_Items1.Add("Item 2",null,null);  
var_Items1.Add("Item 3",null,null);  
var_Items.Add("",null,null).ToString = "[group=0x03][itemspad=4,8,4,8]  
[itemsbg=0x1000000](Item 1,Item 2,Item 3)";  
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Item,com_Item1,com_Items,com_Items1;
    anytype var_Item,var_Item1,var_Items,var_Items1;
    ;

    super();

    exribbon1.VisualAppearance().Add(1,"c:\\exontrol\\images\\normal.ebn");
    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("",COMVariant::createFromInt(2)); com_Item =
var_Item;
    com_Item.GroupPopup(3/*exNoGroupPopupFrame | exGroupPopup*/);
    var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
    com_Items1.BackColor(0x1000000);
    com_Items1.Padding("4,8,4,8");
    com_Items1.Add("Item 1");
    com_Items1.Add("Item 2");
    com_Items1.Add("Item 3");
    var_Item1 = COM::createFromObject(com_Items.Add("")); com_Item1 =
var_Item1;
    com_Item1.ToString("[group=0x03][itemspad=4,8,4,8][itemsbg=0x1000000]
(Item 1,Item 2,Item 3)");
    exribbon1.Refresh();
}
```

Delphi 8 (.NET only)

```
with AxRibbon1 do
begin
    VisualAppearance.Add(1,'c:\\exontrol\\images\\normal.ebn');
    with Items do
    begin
        with Add('',TObject(2),Nil) do
```

```

begin
    GroupPopup :=
Integer(EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib.GroupPopupEnum.exGroupPopup);
    with Items do
        begin
            BackColor := $1000000;
            Padding := '4,8,4,8';
            Add('Item 1',Nil,Nil);
            Add('Item 2',Nil,Nil);
            Add('Item 3',Nil,Nil);
        end;
    end;
    Add('',Nil,Nil).ToString := '[group=0x03][itemspad=4,8,4,8]
[itemsbg=0x1000000](Item 1,Item 2,Item 3)';
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    VisualAppearance.Add(1,'c:\exontrol\images\normal.ebn');
    with Items do
        begin
            with Add('',OleVariant(2),Null) do
                begin
                    GroupPopup := Integer(EXRIBBONLib_TLB.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib_TLB.exGroupPopup);
                    with Items do
                        begin
                            BackColor := $1000000;
                            Padding := '4,8,4,8';
                            Add('Item 1',Null,Null);
                            Add('Item 2',Null,Null);
                            Add('Item 3',Null,Null);

```

```

        end;
    end;
    Add(',Null,Null).ToString := '[group=0x03][itemspad=4,8,4,8]
[itemsbg=0x1000000](Item 1,Item 2,Item 3)';
    end;
    Refresh();
end

```

VFP

```

with thisform.Ribbon1
    .VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")
    with .Items
        with .Add("",2)
            .GroupPopup = 3 && GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
            with .Items
                .BackColor = 0x1000000
                .Padding = "4,8,4,8"
                .Add("Item 1")
                .Add("Item 2")
                .Add("Item 3")
            endwith
        endwith
        .Add("").ToString = "[group=0x03][itemspad=4,8,4,8][itemsbg=0x1000000](Item
1,Item 2,Item 3)"
    endwith
    .Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Items,var_Items1

oRibbon = form.Activex1.nativeObject
oRibbon.VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")
var_Items = oRibbon.Items
    var_Item = var_Items.Add("",2)

```

```

var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
var_Items1 = var_Item.Items
    var_Items1.BackColor = 0x1000000
    var_Items1.Padding = "4,8,4,8"
    var_Items1.Add("Item 1")
    var_Items1.Add("Item 2")
    var_Items1.Add("Item 3")

// var_Items.Add("").ToString = "[group=0x03][itemspad=4,8,4,8]
[itemsbg=0x1000000](Item 1,Item 2,Item 3)"
var_Item1 = var_Items.Add("")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.ToString = "[group=0x03][itemspad=4,8,4,8]
[itemsbg=0x1000000](Item 1,Item 2,Item 3)"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P
Dim var_Items1 as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
oRibbon.VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")
var_Items = oRibbon.Items
    var_Item = var_Items.Add("",2)
    var_Item.GroupPopup = 3 'exNoGroupPopupFrame + exGroupPopup
    var_Items1 = var_Item.Items
        var_Items1.BackColor = 16777216
        var_Items1.Padding = "4,8,4,8"
        var_Items1.Add("Item 1")
        var_Items1.Add("Item 2")

```



```

var_Items1.Add("Item 3")
' var_Items.Add("").ToString = "[group=0x03][itemspad=4,8,4,8]
[itemsbg=0x1000000](Item 1,Item 2,Item 3)"
var_Item1 = var_Items.Add("")
oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1
oRibbon.Template = "var_Item1.ToString = \"[group=0x03][itemspad=4,8,4,8]
[itemsbg=0x1000000](Item 1,Item 2,Item 3)\"

oRibbon.Refresh()

```

Visual Objects

```

local var_Item as Item
local var_Items,var_Items1 as Items

oDCOCX_Exontrol1:VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")
var_Items := oDCOCX_Exontrol1:Items
var_Item := var_Items.Add("",2,nil)
var_Item:GroupPopup := exNoGroupPopupFrame | exGroupPopup
var_Items1 := var_Item:Items
var_Items1:BackColor := 0x1000000
var_Items1:Padding := "4,8,4,8"
var_Items1.Add("Item 1",nil,nil)
var_Items1.Add("Item 2",nil,nil)
var_Items1.Add("Item 3",nil,nil)
var_Items.Add("",nil,nil):ToString := "[group=0x03][itemspad=4,8,4,8]
[itemsbg=0x1000000](Item 1,Item 2,Item 3)"
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```

OleObject oRibbon,var_Item,var_Items,var_Items1

oRibbon = ole_1.Object
oRibbon:VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")

```

```

var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
var_Items1 = var_Item.Items
var_Items1.BackColor = 16777216 /*0x1000000*/
var_Items1.Padding = "4,8,4,8"
var_Items1.Add("Item 1")
var_Items1.Add("Item 2")
var_Items1.Add("Item 3")
var_Items.Add("").ToString = "[group=0x03][itemspad=4,8,4,8]
[itemsbg=0x1000000](Item 1,Item 2,Item 3)"
oRibbon.Refresh()

```

Visual DataFlex

```

Procedure OnCreate
  Forward Send OnCreate
  Variant voAppearance
  Get ComVisualAppearance to voAppearance
  Handle hoAppearance
  Get Create (RefClass(cComAppearance)) to hoAppearance
  Set pvComObject of hoAppearance to voAppearance
  Get ComAdd of hoAppearance 1 "c:\exontrol\images\normal.ebn" to Nothing
  Send Destroy to hoAppearance
  Variant voltems
  Get ComItems to voltems
  Handle holtems
  Get Create (RefClass(cComItems)) to holtems
  Set pvComObject of holtems to voltems
  Variant voltem
  Get ComAdd of holtems "" 2 Nothing to voltem
  Handle holtem
  Get Create (RefClass(cComItem)) to holtem
  Set pvComObject of holtem to voltem
  Set ComGroupPopup of holtem to (OLEexNoGroupPopupFrame +
OLEexGroupPopup)

```

```

Variant voltems1
Get ComItems of holtem to voltems1
Handle holtems1
Get Create (RefClass(cComItems)) to holtems1
Set pvComObject of holtems1 to voltems1
    Set ComBackColor of holtems1 to |CI$1000000
    Set ComPadding of holtems1 to "4,8,4,8"
    Get ComAdd of holtems1 "Item 1" Nothing Nothing to Nothing
    Get ComAdd of holtems1 "Item 2" Nothing Nothing to Nothing
    Get ComAdd of holtems1 "Item 3" Nothing Nothing to Nothing
Send Destroy to holtems1
Send Destroy to holtem
Variant voltem1
Get ComAdd of holtems "" Nothing Nothing to voltem1
Handle holtem1
Get Create (RefClass(cComItem)) to holtem1
Set pvComObject of holtem1 to voltem1
    Set ComToString of holtem1 to "[group=0x03][itemspad=4,8,4,8]
[itemzburg=0x1000000](Item 1,Item 2,Item 3)"
Send Destroy to holtem1
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem
    LOCAL oltems,oltems1
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )

```

```

oForm:drawingArea:clipChildren := .T.
oForm:create( ,, {100,100}, {640,480},,, .F. )
oForm:close := {|| PostAppEvent( xbeP_Quit )}

oRibbon := XbpActiveXControl():new( oForm:drawingArea )
oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
oRibbon:create(,, {10,60},{610,370} )

oRibbon:VisualAppearance():Add(1,"c:\exontrol\images\normal.ebn")
oltems := oRibbon:Items()
oltem := oltems:Add("",2)
oltem:GroupPopup := 3/*exNoGroupPopupFrame+exGroupPopup*/
oltems1 := oltem:Items()
oltems1:SetProperty("BackColor",0x1000000)
oltems1:Padding := "4,8,4,8"
oltems1:Add("Item 1")
oltems1:Add("Item 2")
oltems1:Add("Item 3")
oltems:Add(""):ToString := "[group=0x03][itemspad=4,8,4,8]
[itemsbg=0x1000000](Item 1,Item 2,Item 3)"
oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

property Items.BackgroundExt as String

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

Type	Description
String	A String expression (" EBN String Format ") that defines the layout of the UI to be applied on the items' background. The syntax of EBN String Format in BNF notation is shown bellow. <i>You can use the EBN's Builder of eXButton/COM control to define visually the EBN String Format.</i>

By default, the BackgroundExt property is empty. Using the BackgroundExt property you have unlimited options to show any HTML text, images, colors, EBNs, patterns, frames anywhere on the items' background. *For instance, let's say you need to display **more** colors on the items' background, or just want to display an **additional** caption or image to a specified location on the items' background.* The EBN String Format defines the parts of the EBN to be applied on the items' background. The [EBN](#) is a set of UI elements that are built as a tree where each element is anchored to its parent element. The BackgroundExt property is applied right after setting the object's bgcolor, and before drawing the default object's captions, icons or pictures.

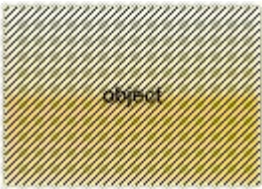
In the following screen shot the Views, Clipboard, Font and Speakers are shown using the BackgroundExt property:



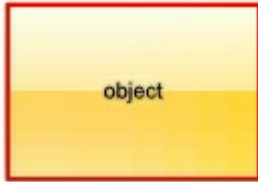
as "bottom[2],bottom[16,text=`</fgcolor><fgcolor 6D6AAA>Views</fgcolor><fgcolor A0A0A0>`,align=0x21]", shows the Views aligned to the bottom, with a different foreground color.

Easy samples:

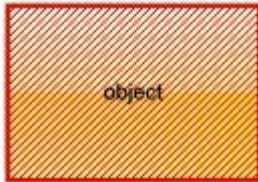
- "[pattern=6]", shows the BDiagonal pattern on the object's background.



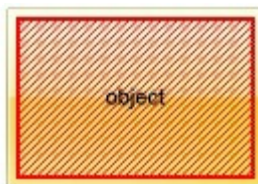
- "[frame=RGB(255,0,0),framethick]", draws a red thick-border around the object.



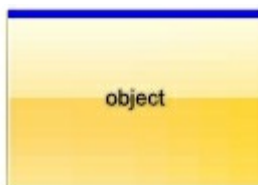
- "[frame=RGB(255,0,0),framethick,pattern=6,patterncolor=RGB(255,0,0)]", draws a red thick-border around the object, with a patten inside.



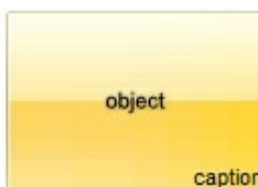
- "[[patterncolor=RGB(255,0,0)]
(none[(4,4,100%-8,100%-8),pattern=0x006,patterncolor=RGB(255,0,0),frame=RGB(255,0,0),framethick])]", draws a red thick-border around the object, with a patten inside, with a 4-pixels wide padding:



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



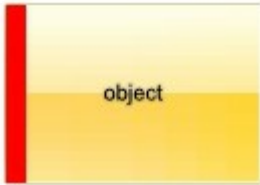
- "[text=`caption`,align=0x22]", shows the caption string aligned to the bottom-right side of the object's background.



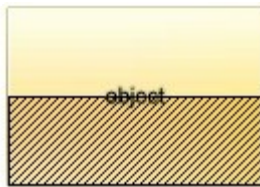
- "[text=`flag`,align=0x11]" shows the flag picture and the sweden string aligned to the bottom side of the object.



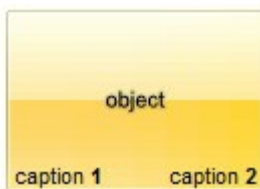
- "left[10,back=RGB(255,0,0)]", draws a red line on the left side of the object's background, of 10-pixels wide.



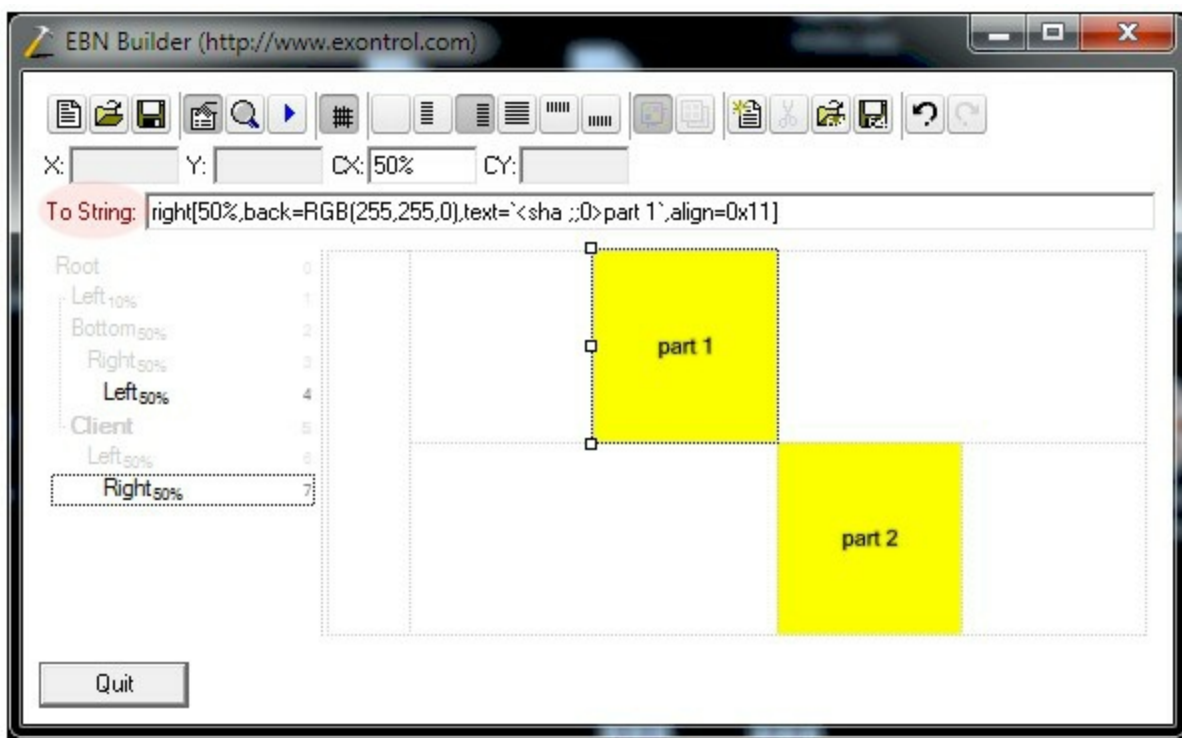
- "bottom[50%,pattern=6,frame]", shows the BDiagonal pattern with a border around on the lower-half part of the object's background.



- "root[text=`caption 2` ,align=0x22](client[text=`caption 1` ,align=0x20])", shows the caption 1 aligned to the bottom-left side, and the caption 2 to the bottom-right side



The Exontrol's [eXButton](#) WYSWYG Builder helps you to generate or view the EBN String Format, in the **To String** field as shown in the following screen shot:



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

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

```

<EBN> ::= <elements> | <root> "(" [<elements>] ")"
<elements> ::= <element> [ "," <elements> ]
<root> ::= "root" [ <attributes> ] | [ <attributes> ]
<element> ::= <anchor> [ <attributes> ] [ "(" [<elements>] ")" ]
<anchor> ::= "none" | "left" | "right" | "client" | "top" | "bottom"
<attributes> ::= "[" [<client> "," <attribute> [ "," <attributes> ] "]"
<client> ::= <expression> | <expression> "," <expression> "," <expression> ","
<expression>
<expression> ::= <number> | <number> "%"
<attribute> ::= <backcolor> | <text> | <wordwrap> | <align> | <pattern> |
<patterncolor> | <frame> | <framethick> | <data> | <others>
<equal> ::= "="
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<decimal> ::= <digit> <decimal>
<hexadigit> ::= <digit> | "A" | "B" | "C" | "D" | "E" | "F"
<hexa> ::= <hexadigit> <hexa>
<number> ::= <decimal> | "0x" <hexa>
<color> ::= <rgbcolor> | number
<rgbcolor> ::= "RGB" "(" <number> "," <number> "," <number> ")"

```


`<string> ::= "`" <characters> "`" | "'" <characters> "'" | " <characters> "`
`<characters> ::= <char>|<characters>`
`<char> ::= <any_character_excepts_null>`
`<backcolor> ::= "back" <equal> <color>`
`<text> ::= "text" <equal> <string>`
`<align> ::= "align" <equal> <number>`
`<pattern> ::= "pattern" <equal> <number>`
`<patterncolor> ::= "patterncolor" <equal> <color>`
`<frame> ::= "frame" <equal> <color>`
`<data> ::= "data" <equal> <number> | <string>`
`<framethick> ::= "framethick"`
`<wordwrap> ::= "wordwrap"`

Others like: pic, stretch, hstretch, vstretch, transparent, from, to are reserved for future use only.

method Items.Clear ()

Removes all objects in a collection.

Type	Description
	Use the Clear method to clear all elements/items in the collection. The Remove method removes an item giving its identifier.

property Items.Count as Long

Returns the number of objects in a collection.

Type	Description
Long	A Long expression that specifies the number of Item objects in the collection.

The Count property specifies the the number of [Item](#) objects in the collection. The [Add](#) method adds a new item to the Items collection, while the [Remove](#) method removes an item giving its identifier. Use the [Clear](#) method to clear all elements/items in the collection.

property Items.HotBackColor as Color

Specifies the hot background color of the items (when the cursor hovers the items).

Type	Description
Color	A Color expression that specifies the items' background color, when the cursor hovers it. 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.

The HotBackColor property specifies the background color for items when cursor hovers it. The [BackColor](#) property specifies the solid color / visual appearance to be shown on the items' background (inside borders). The [BackgroundExt](#) property indicates additional colors, text, images that can be displayed on the items's background using the EBN string format. The [PopupAppearance](#) property specifies the visual appearance of the items (including the margins/borders). The [Padding](#) property specifies the padding (space between the menu border and the item content) to display the items.

How can I change the visual appearance for items, when cursor hovers it (hot,EBN)?

VBA (MS Access, Excell...)

With Ribbon1

```
.VisualAppearance.Add 1,"c:\exontrol\images\normal.ebn"
```

With .Items

```
With .Add("",2)
```

```
.GroupPopup = 3 ' GroupPopupEnum.exNoGroupPopupFrame Or  
GroupPopupEnum.exGroupPopup
```

With .Items

```
.HotBackColor = &H1000000
```

```
.Padding = "4,8,4,8"
```

```
.Add "Item 1"
```

```
.Add "Item 2"
```

```
.Add "Item 3"
```

End With

End With

```
.Add("").ToString = "[group=0x03][itemspad=4,8,4,8][itemsbghot=0x1000000]
```

```
(Item 1,Item 2,Item 3)"
```

```
End With
```

```
.Refresh
```

```
End With
```

VB6

```
With Ribbon1
```

```
.VisualAppearance.Add 1,"c:\exontrol\images\normal.ebn"
```

```
With .Items
```

```
With .Add("",2)
```

```
.GroupPopup = GroupPopupEnum.exNoGroupPopupFrame Or
```

```
GroupPopupEnum.exGroupPopup
```

```
With .Items
```

```
.HotBackColor = &H1000000
```

```
.Padding = "4,8,4,8"
```

```
.Add "Item 1"
```

```
.Add "Item 2"
```

```
.Add "Item 3"
```

```
End With
```

```
End With
```

```
.Add("").ToString = "[group=0x03][itemspad=4,8,4,8][itemsbgshot=0x1000000]
```

```
(Item 1,Item 2,Item 3)"
```

```
End With
```

```
.Refresh
```

```
End With
```

VB.NET

```
With Exribbon1
```

```
.VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")
```

```
With .Items
```

```
With .Add("",2)
```

```
.GroupPopup =
```

```
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or
```

```
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup
```

```
With .Items
```

```
.HotBackColor32 = &H1000000
```

```

        .Padding = "4,8,4,8"
        .Add("Item 1")
        .Add("Item 2")
        .Add("Item 3")
    End With
End With
.Add("").ToString = "[group=0x03][itemspad=4,8,4,8][itemsbghot=0x1000000]
(Item 1,Item 2,Item 3)"
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    .VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")
    With .Items
        With .Add("",2)
            .GroupPopup = EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or
EXRIBBONLib.GroupPopupEnum.exGroupPopup
            With .Items
                .HotBackColor = &H1000000
                .Padding = "4,8,4,8"
                .Add("Item 1")
                .Add("Item 2")
                .Add("Item 3")
            End With
        End With
    End With
    .Add("").ToString = "[group=0x03][itemspad=4,8,4,8][itemsbghot=0x1000000]
(Item 1,Item 2,Item 3)"
End With
.Refresh()
End With

```

C++

```

/*
Copy and paste the following directives to your header file as

```

it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control Library'

```
#import <ExRibbon.dll>
using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
spRibbon1->GetVisualAppearance()->Add(1,"c:\\exontrol\\images\\normal.ebn");
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"",long(2),vtMissing);
    var_Item-
>PutGroupPopup(EXRIBBONLib::GroupPopupEnum(EXRIBBONLib::exNoGroupPopupFrame
| EXRIBBONLib::exGroupPopup));
    EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();
    var_Items1->PutHotBackColor(0x1000000);
    var_Items1->PutPadding(L"4,8,4,8");
    var_Items1->Add(L"Item 1",vtMissing,vtMissing);
    var_Items1->Add(L"Item 2",vtMissing,vtMissing);
    var_Items1->Add(L"Item 3",vtMissing,vtMissing);
    var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"[group=0x03]
[itemspad=4,8,4,8][itemsbghot=0x1000000](Item 1,Item 2,Item 3)");
spRibbon1->Refresh();
```

C++ Builder

```
Ribbon1->VisualAppearance-
>Add(1,TVariant("c:\\exontrol\\images\\normal.ebn"));
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    Exribbonlib_tlb::IItemPtr var_Item = var_Items->Add(L"",TVariant(2),TNoParam());
    var_Item->GroupPopup =
Exribbonlib_tlb::GroupPopupEnum::exNoGroupPopupFrame |
Exribbonlib_tlb::GroupPopupEnum::exGroupPopup;
    Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;
    var_Items1->HotBackColor = 0x1000000;
    var_Items1->Padding = L"4,8,4,8";
```

```

var_Items1->Add(L"Item 1",TNoParam(),TNoParam());
var_Items1->Add(L"Item 2",TNoParam(),TNoParam());
var_Items1->Add(L"Item 3",TNoParam(),TNoParam());
var_Items->Add(L"",TNoParam(),TNoParam())-> ToString = L"[group=0x03]
[itemspad=4,8,4,8][itemsbghot=0x1000000](Item 1,Item 2,Item 3)";
Ribbon1->Refresh();

```

C#

```

exribbon1.VisualAppearance.Add(1,"c:\\exontrol\\images\\normal.ebn");
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
var_Item.GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup;
exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
var_Items1.HotBackColor32 = 0x1000000;
var_Items1.Padding = "4,8,4,8";
var_Items1.Add("Item 1",null,null);
var_Items1.Add("Item 2",null,null);
var_Items1.Add("Item 3",null,null);
var_Items.Add("",null,null).ToString = "[group=0x03][itemspad=4,8,4,8]
[itemsbghot=0x1000000](Item 1,Item 2,Item 3)";
exribbon1.Refresh();

```

JScrip/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JavaScript">
function Init()
{
Ribbon1.VisualAppearance.Add(1,"c:\\exontrol\\images\\normal.ebn");
var var_Items = Ribbon1.Items;

```



```

var var_Item = var_Items.Add("",2,null);
var_Item.GroupPopup = 3;
var var_Items1 = var_Item.Items;
var_Items1.HotBackColor = 16777216;
var_Items1.Padding = "4,8,4,8";
var_Items1.Add("Item 1",null,null);
var_Items1.Add("Item 2",null,null);
var_Items1.Add("Item 3",null,null);
var_Items.Add("",null,null).ToString = "[group=0x03][itemspad=4,8,4,8]
[itemsbghot=0x1000000](Item 1,Item 2,Item 3)";
Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
With Ribbon1
.VisualAppearance.Add 1,"c:\exontrol\images\normal.ebn"
With .Items
With .Add("",2)
.GroupPopup = 3 ' GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
With .Items
.HotBackColor = &H1000000
.Padding = "4,8,4,8"
.Add "Item 1"
.Add "Item 2"
.Add "Item 3"
End With
End With

```

```

End With
.Add("").ToString = "[group=0x03][itemspad=4,8,4,8]
[itemsbghot=0x1000000](Item 1,Item 2,Item 3)"
End With
.Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

axRibbon1.VisualAppearance.Add(1,"c:\\exontrol\\images\\normal.ebn");
EXRIBBONLib.Items var_Items = axRibbon1.Items;
EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
var_Item.GroupPopup =
EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
EXRIBBONLib.GroupPopupEnum.exGroupPopup;
EXRIBBONLib.Items var_Items1 = var_Item.Items;
var_Items1.HotBackColor = 0x1000000;
var_Items1.Padding = "4,8,4,8";
var_Items1.Add("Item 1",null,null);
var_Items1.Add("Item 2",null,null);
var_Items1.Add("Item 3",null,null);
var_Items.Add("",null,null).ToString = "[group=0x03][itemspad=4,8,4,8]
[itemsbghot=0x1000000](Item 1,Item 2,Item 3)";
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
COM com_Item,com_Item1,com_Items,com_Items1;
anytype var_Item,var_Item1,var_Items,var_Items1;
;

```

```

super();

exribbon1.VisualAppearance.Add(1,"c:\\exontrol\\images\\normal.ebn");
var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("",COMVariant::createFromInt(2)); com_Item =
var_Item;
    com_Item.GroupPopup(3/*exNoGroupPopupFrame | exGroupPopup*/);
    var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
        com_Items1.HotBackColor(0x1000000);
        com_Items1.Padding("4,8,4,8");
        com_Items1.Add("Item 1");
        com_Items1.Add("Item 2");
        com_Items1.Add("Item 3");
    var_Item1 = COM::createFromObject(com_Items.Add("")); com_Item1 =
var_Item1;
    com_Item1.ToString("[group=0x03][itemspad=4,8,4,8][itemsbgshot=0x1000000]
(Item 1,Item 2,Item 3)");
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    VisualAppearance.Add(1,'c:\\exontrol\\images\\normal.ebn');
    with Items do
    begin
        with Add('',TObject(2),Nil) do
        begin
            GroupPopup :=
Integer(EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib.GroupPopupEnum.exGroupPopup);
            with Items do
            begin
                HotBackColor := $1000000;
                Padding := '4,8,4,8';
                Add('Item 1',Nil,Nil);
            end
        end
    end
end

```

```

        Add('Item 2',Nil,Nil);
        Add('Item 3',Nil,Nil);
    end;
end;
Add('',Nil,Nil).ToString := '[group=0x03][itemspad=4,8,4,8]
[itemsbghot=0x1000000](Item 1,Item 2,Item 3)';
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
    VisualAppearance.Add(1,'c:\exontrol\images\normal.ebn');
    with Items do
    begin
        with Add('',OleVariant(2),Null) do
        begin
            GroupPopup := Integer(EXRIBBONLib_TLB.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib_TLB.exGroupPopup);
            with Items do
            begin
                HotBackColor := $1000000;
                Padding := '4,8,4,8';
                Add('Item 1',Null,Null);
                Add('Item 2',Null,Null);
                Add('Item 3',Null,Null);
            end;
        end;
    end;
    Add('',Null,Null).ToString := '[group=0x03][itemspad=4,8,4,8]
[itemsbghot=0x1000000](Item 1,Item 2,Item 3)';
end;
Refresh();
end

```

with thisform.Ribbon1

.VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")

with .Items

with .Add("",2)

.GroupPopup = 3 && GroupPopupEnum.exNoGroupPopupFrame Or

GroupPopupEnum.exGroupPopup

with .Items

.HotBackColor = 0x1000000

.Padding = "4,8,4,8"

.Add("Item 1")

.Add("Item 2")

.Add("Item 3")

endwith

endwith

.Add("").ToString = "[group=0x03][itemspad=4,8,4,8][itemsbghot=0x1000000]
(Item 1,Item 2,Item 3)"

endwith

.Refresh

endwith

dBASE Plus

local oRibbon,var_Item,var_Item1,var_Items,var_Items1

oRibbon = form.ActiveX1.nativeObject

oRibbon.**VisualAppearance**.Add(1,"c:\exontrol\images\normal.ebn")

var_Items = oRibbon.Items

var_Item = var_Items.Add("",2)

var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/

var_Items1 = var_Item.Items

var_Items1.**HotBackColor** = 0x1000000

var_Items1.Padding = "4,8,4,8"

var_Items1.Add("Item 1")

var_Items1.Add("Item 2")

var_Items1.Add("Item 3")

// var_Items.Add("").ToString = "[group=0x03][itemspad=4,8,4,8]
[itemsbghot=0x1000000](Item 1,Item 2,Item 3)"

```

var_Item1 = var_Items.Add("")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.ToString = "[group=0x03][itemspad=4,8,4,8]
[itemsbghot=0x1000000](Item 1,Item 2,Item 3)"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P
Dim var_Items1 as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
oRibbon.VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 3 'exNoGroupPopupFrame + exGroupPopup
var_Items1 = var_Item.Items
var_Items1.HotBackColor = 16777216
var_Items1.Padding = "4,8,4,8"
var_Items1.Add("Item 1")
var_Items1.Add("Item 2")
var_Items1.Add("Item 3")
'var_Items.Add("").ToString = "[group=0x03][itemspad=4,8,4,8]
[itemsbghot=0x1000000](Item 1,Item 2,Item 3)"
var_Item1 = var_Items.Add("")
oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1
oRibbon.Template = "var_Item1.ToString = \"[group=0x03][itemspad=4,8,4,8]
[itemsbghot=0x1000000](Item 1,Item 2,Item 3)\\"

```

```
oRibbon.Refresh()
```

Visual Objects

```
local var_Item as Item
local var_Items,var_Items1 as Items

oDCOCX_Exontrol1:VisualAppearance:Add(1,"c:\exontrol\images\normal.ebn")
var_Items := oDCOCX_Exontrol1:Items
var_Item := var_Items:Add("",2,nil)
var_Item:GroupPopup := exNoGroupPopupFrame | exGroupPopup
var_Items1 := var_Item:Items
var_Items1:HotBackColor := 0x1000000
var_Items1:Padding := "4,8,4,8"
var_Items1:Add("Item 1",nil,nil)
var_Items1:Add("Item 2",nil,nil)
var_Items1:Add("Item 3",nil,nil)
var_Items:Add("",nil,nil):ToString := "[group=0x03][itemspad=4,8,4,8]
[itemsbghot=0x1000000](Item 1,Item 2,Item 3)"
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Items,var_Items1

oRibbon = ole_1.Object
oRibbon.VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")
var_Items = oRibbon.Items
var_Item = var_Items.Add("",2)
var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
var_Items1 = var_Item.Items
var_Items1.HotBackColor = 16777216 /*0x1000000*/
var_Items1.Padding = "4,8,4,8"
var_Items1.Add("Item 1")
var_Items1.Add("Item 2")
var_Items1.Add("Item 3")
```

```
var_Items.Add("").ToString = "[group=0x03][itemspad=4,8,4,8]  
[itemsbghot=0x1000000](Item 1,Item 2,Item 3)"  
oRibbon.Refresh()
```

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Variant voAppearance

Get **ComVisualAppearance** to voAppearance

Handle hoAppearance

Get Create (RefClass(cComAppearance)) to hoAppearance

Set pvComObject of hoAppearance to voAppearance

Get ComAdd of hoAppearance 1 "c:\exontrol\images\normal.ebn" to Nothing

Send Destroy to hoAppearance

Variant voltems

Get Comltems to voltems

Handle holtems

Get Create (RefClass(cComltems)) to holtems

Set pvComObject of holtems to voltems

Variant voltem

Get ComAdd of holtems "" 2 Nothing to voltem

Handle holtem

Get Create (RefClass(cComltem)) to holtem

Set pvComObject of holtem to voltem

Set ComGroupPopup of holtem to (OLEexNoGroupPopupFrame +
OLEexGroupPopup)

Variant voltems1

Get Comltems of holtem to voltems1

Handle holtems1

Get Create (RefClass(cComltems)) to holtems1

Set pvComObject of holtems1 to voltems1

Set **ComHotBackColor** of holtems1 to |C|\$1000000

Set ComPadding of holtems1 to "4,8,4,8"

Get ComAdd of holtems1 "Item 1" Nothing Nothing to Nothing

Get ComAdd of holtems1 "Item 2" Nothing Nothing to Nothing


```

        Get ComAdd of holtems1 "Item 3" Nothing Nothing to Nothing
    Send Destroy to holtems1
Send Destroy to holtem
Variant voltem1
Get ComAdd of holtems "" Nothing Nothing to voltem1
Handle holtem1
Get Create (RefClass(cComItem)) to holtem1
Set pvComObject of holtem1 to voltem1
    Set ComToString of holtem1 to "[group=0x03][itemspad=4,8,4,8]
[itemsbghot=0x1000000](Item 1,Item 2,Item 3)"
    Send Destroy to holtem1
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem
    LOCAL oltems,oltems1
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

```

```

oRibbon:VisualAppearance():Add(1,"c:\exontrol\images\normal.ebn")
oltems := oRibbon:Items()
oltem := oltems:Add("",2)
oltem:GroupPopup := 3/*exNoGroupPopupFrame+exGroupPopup*/
oltems1 := oltem:Items()
oltems1:SetProperty("HotBackColor",0x1000000)
oltems1:Padding := "4,8,4,8"
oltems1:Add("Item 1")
oltems1:Add("Item 2")
oltems1:Add("Item 3")
oltems:Add(""):ToString := "[group=0x03][itemspad=4,8,4,8]
[itemsbghot=0x1000000](Item 1,Item 2,Item 3)"
oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

property Items.Item (ID as Variant) as Item

Returns a specific Item object giving its identifier.

Type	Description
ID as Variant	A Long expression that specifies the identifier of the item being requested or a String expression that specifies the caption of the item being requested.
Item	An Item object with associated identifier.

The Item property looks in the Items collection for the item with the specified identifier or caption. You can use the [Item](#) property of the ribbon control to recursively search for an item giving its identifier or caption. The [ID](#) property of the Item object specifies the identifier of the item. The [Caption](#) property of the Item object specifies the caption of the item. The Item property gets the first Item object being found, if multiple objects with the same identifier are found, or Nothing, if no item with associated identifier is found.

property Items.Padding as String

Specifies the padding (space between the menu border and the item content) to display the items.

Type	Description
String	A string expression that indicates a list of 4 positive numbers separated by comma characters, which indicates the distance in pixels from margin to client, in the following format: left, top, right, bottom.

By default, the Padding property is empty (0,0,0,0). The Padding property specifies the padding (space between the menu border and the item content) to display the items. The [BackgroundExt](#) property indicates additional colors, text, images that can be displayed on the items's background using the EBN string format. When using EBN appearance, using the [PopupAppearance](#), [LocalAppearance](#) or [Appearance](#), the distance between margins/borders and items client area is indicated by the client object of the skin/ebn object. The [Padding](#) property specifies the padding for a particular item.

The following screen shot shows the control with no padding:



The following screen shot shows the control with padding 16, 16, 16, 16:



property Items.PopupAppearance as AppearanceEnum

Retrieves or sets the popup's appearance.

Type	Description
AppearanceEnum	<p>A AppearanceEnum expression that specifies the popup's frame appearance, or a color expression whose last 7 bits in the high significant byte of the value indicates the index of the skin in the Appearance collection, being displayed as control's borders. For instance, if the Appearance = 0x1000000, indicates that the first skin object in the Appearance collection defines the control's border. <i>The Client object in the skin, defines the client area of the control. 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</i></p>

By default, the PopupAppearance property is specified by the control's [Appearance](#) property. The PopupAppearance specifies a different visual appearance for the current submenu. The [SubMenu](#) property determines the items to be displayed on the popup item. The [BackColor](#) property specifies the solid color / visual appearance to be shown on the items' background (inside borders). The [BackgroundExt](#) property indicates additional colors, text, images that can be displayed on the items's background using the EBN string format. When using EBN appearance, using the PopupAppearance, [LocalAppearance](#) or [Appearance](#), the distance between margins/borders and items client area is indicated by the client object of the skin/ebn object.

The appearance of the popup is determined by the following:

- PopupAppearance, specifies the visual appearance of the current sub-menu.
- [LocalAppearance](#), determines the visual appearance of the popup, if it is local ([ShowLocalPopup](#) property)
- [Appearance](#), specifies the general visual appearance of the popup items.

The following screen shot shows the sub-menu with different appearances:



(single appearance)



(shadow appearance)



(ebn appearance)



(ebn appearance)

method Items.Remove (ID as Variant)

Removes a specific member from the collection.

Type	Description
ID as Variant	A Long expression that specifies the item to be removed. A String expression that specifies the caption of the Item to be removed

The Remove method removes an individual Item object giving its identifier or caption. The [Visible](#) property specifies whether the item is visible or hidden.

property Items.SortOrder as SubMenuSortOrderEnum

Sorts the items in the submenu.

Type	Description
SubMenuSortOrderEnum	A SubMenuSortOrderEnum expression that specifies the way the submenu displays its items.

By default, the SortOrder property is exSubMenuUnsorted, which indicates that the items are displayed on the submenu as they were added. Use the SortOrder property to sort the items to be displayed on the sub menu.

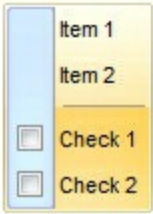
property Items.ToString as String

Loads or saves the Items collection using string representation.

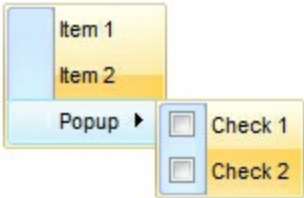
Type	Description
String	A String expression that specifies the items to be added. The list of items is separated by , (comma) character, while sub-menus are include between () parenthesis. The [] brackets indicates the options to be applied on the item

The ToString property loads or saves the control items from a string. The [Add](#) method adds a new item to the Items collection. The [Remove](#) method removes a specified item.

For instance, the *"Item 1,Item 2,[sep],Check 1[chk],Check 2[chk]"*, generates the following screen shot:



For instance, the *"Item 1,Item 2,Popup(Check 1[chk],Check 2[chk])"*, generates the following screen shot:



For instance, the *"Calendar[id=20][img=0],MSChart[id=30],Record[id=40],Slider[id=50],Radio 1[id=100][typ=2][edit=],Radio 2[id=101][typ=2][edit=],Radio 3[id=102][typ=2][edit=],ComboBox[id=90]"*, generates the following screen shot:



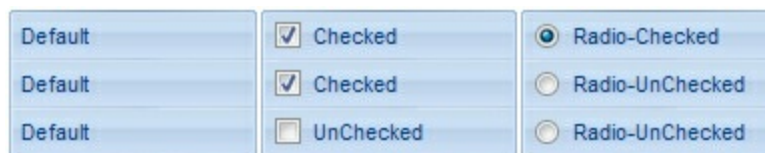
The ToString syntax in BNF notation:

```
<ToString> ::= <ITEMS>
<ITEMS> ::= <ITEM>["("<ITEMS>")"][","<ITEMS>]
<ITEM> ::= <CAPTION>[<OPTIONS>]
<OPTIONS> ::= "["<OPTION>"]"["["<OPTIONS>"]"]
<OPTION> ::= <PROPERTY>["="<VALUE>]
<PROPERTY> ::= "img" | "himg" | "sep" | "id" | "typ" | "group" | "chk" | "button" | "align" |
"spchk" | "show" | "rad" | "dis" | "showdis" | "bld" | "itl" | "stk" | "und" | "bg" | "fg" | "edittype" |
"edit" | "mask" | "border" | "editwidth" | "captionwidth" | "height" | "grp" | "tfi" | "ttp" | "min" |
"max" | "tick" | "freq" | "ticklabel" | "small" | "large" | "spin" | "ettp" | "float" | "close" | "local" |
"popupapp" | "itemspad" | "itemsbg" | "itemsbghot" | "itemsbgext" | "visible" | "tab" | "pad" |
"bghot" | "bgssel" | "bgsselhot" | "arrow" | "popupalign" | "popupoffset" | "popupat" | "hid"
```

where the <CAPTION> is the HTML caption to be shown on the context menu item. The <VALUE> indicates the value of giving property.

- id=<VALUE>, where <VALUE> is an integer expression, that indicates the identifier of the item.
- bg=<VALUE>, specifies the item's background color, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or an integer expression to that refers an EBN object.
- bghot=<VALUE>, specifies the item's background color, while the cursor hovers the item, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or an integer expression to that refers an EBN object.
- bgssel=<VALUE>, specifies the item's background color, while the item is checked/selected, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or an integer expression to that refers an EBN object.
- bgsselhot=<VALUE>, specifies the item's background color, while the item is checked/selected and the cursor hovers it, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or an integer expression to that refers an EBN object.
- fg=<VALUE>, specifies the item's foreground color, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or a integer expression.
- sep, specifies an separator item
- dis, specifies a disabled item
- hid, specifies a hidden item
- showdis=<VALUE>, where <VALUE> could be **0** for regular or **not zero** to specify whether the item shows as disabled, but it is still enabled
- bld, specifies that the item appears in bold

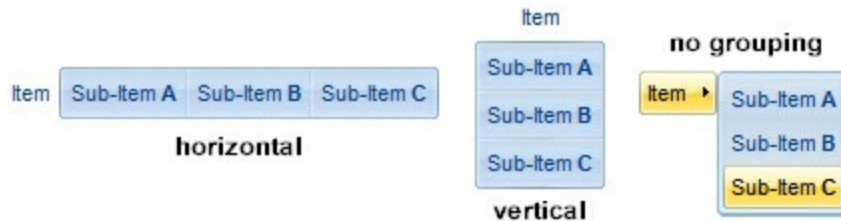
- `itl`, specifies that the item appears in italics
- `stk`, specifies that the item appears as strikethrough
- `und`, specifies that the item is underlined
- `align=<VALUE>`, where `<VALUE>` could be one of the following:
 - **0** (left), to align the item's caption to the left
 - **1** (center), to center the item's caption
 - **2** (right), to align the item's caption to the right
- `captionwidth=<VALUE>`, specifies the width to show the HTML caption of the item. where `<VALUE>` could be a integer expression. A negative value indicates that no limitation is applied to the item's caption, so no truncate caption is shown
- `height=<VALUE>`, specifies the height to show the item, where `<VALUE>` could be a positive integer expression
- `pad=<VALUE>`, specifies the padding (space between the menu border and the item content) to display the item. The `<VALUE>` is a list of coordinates such as `left,top,right,bottom`
- `img=<VALUE>`, where `<VALUE>` is an integer expression, that indicates the index of the icon being displayed for the item.
- `himg=<VALUE>`, where `<VALUE>` indicates the key of the picture to be displayed for the item.



- `typ=<VALUE>`, where `<VALUE>` could be one of the following:
 - **0** for default/regular items (no check/radio button is associated with the item),
 - **1** for items that display a check/box (`chk`),
 - **2** to display radio buttons (`rad`)
- `chk[=<VALUE>]`, where `<VALUE>` could be **0** for unchecked, or **not zero** for checked. The `chk` option makes the item to display a check box. If the `<VALUE>` is missing the item still displays an un-checked check box.
- `rad=<VALUE>`, where `<VALUE>` could be **0** for unchecked radio button or **not zero** to for checked radio button. Use the `grp` option to define the group of radio where this button should be associated, If no group of radio buttons is required, the `grp` could be ignored.
- `grp=<VALUE>`, defines the radio group. It should be used when you define more groups of radio buttons. A group of radio buttons means that only one item could be checked at one time. The `rad` option specifies that the item displays a radio button. Use the `grp` option to define the group of radio where this button should be associated, If no group of radio buttons is required, the `grp` could be ignored. The `<VALUE>` could be any integer expression.



- show=<VALUE>, where <VALUE> could be **0** for regular or **not zero** to specify whether the checked item shows as selected
- spchk=<VALUE>, where <VALUE> could be **0** for regular or **not zero** to specify whether the item's sub menu is shown only if the item is checked.



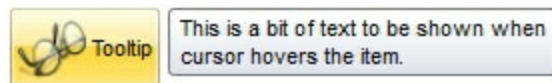
- group=<VALUE>, where <VALUE> could be a bit-or combination (+) of the following values:
 - **0** (exNoGroupPopup), No grouping is performed on the sub-menu, so the sub-items are shown to a float popup,
 - **1** (exGroupPopup), Groups and displays the sub-menu items on the current item, arranged from left to right/horizontally
 - **2** (exNoGroupPopupFrame), Prevents showing the frame around each grouping item.
 - **4** (exGroupPopupCenter), Shows the grouping popup aligned to the center of the current item.
 - **8** (exGroupPopupRight), Shows the grouping popup aligned to the right of the current item.
 - **16** (exGroupPopupEqualWidth), Shows the items that make the group of the same width
 - **32** (exGroupPopupEqualHeight), Shows the items that make the group of the same height
 - **64** (exGroupPopupFrameSolidBox), Shows a solid frame around the grouped items
 - **128** (exGroupPopupFrameThickBox), Shows a solid thick-frame around the grouped items
 - **256** (exGroupPopupVertical), Groups and displays the sub-menu items on the current item, arranged from top to bottom/vertically



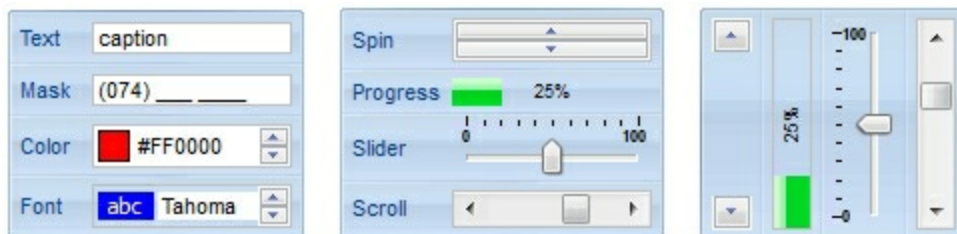
- button=<VALUE>, where <VALUE> could be a bit-or combination (+) of the following

values.

- **0** (exShowAsButtonNone), No button is shown,
- **1** (exShowAsButton), Shows the item as a button
- **2** (exShowAsButtonAutoSize), Fits the button to cover the item's caption instead showing on the entire item
- **17** (exShowAsSelectButton), Shows the item as a select button, which is composed by two-fields, one indicates the default button, while the second field specifies the drop down button that displays the items in the current's sub-menu collection. The drop down button is shown to the right-side of the default button. The item must have a submenu, else no drop down is displayed.
- **273** (exShowAsSelectButtonBottom), Shows the item as a select button, which is composed by two-fields, one indicates the default button, while the second field specifies the drop down button that displays the items in the current's sub-menu collection. The drop down button is shown to the bottom-side of the default button. The item must have a submenu, else no drop down is displayed.



- ttp=<VALUE>, defines the item's tooltip. The <VALUE> could be any HTML string expression. The item's tooltip is shown when the user hovers the item.



- edittype=<VALUE>, associates an edit field to the item, where <VALUE> could be a combination of one or more of the following values:
 - **0** (exlItemDisableEdit), No editor is assigned to the current item.
 - **1** (exlItemEditText), A text-box editor is assigned to the current item.
 - **2** (exlItemEditMask), A masked text-box editor is assigned to the current item.
 - **3** (exlItemEditSlider), A slider editor is assigned to the current item. This can be combined with 1024.
 - **4** (exlItemEditProgress), A progress editor is assigned to the current item. This can be combined with 1024.
 - **5** (exlItemEditScrollBar), A scrollbar editor is assigned to the current item. This can be combined with 1024.
 - **6** (exlItemEditColor), A color editor is assigned to the current item.
 - **7** (exlItemEditFont), A font editor is assigned to the current item.
 - **256** (exlItemEditReadOnly), specifies that the item's editor is shown as disabled. This value could be combined with one of the values from 0 to 7 or 512

- 512 (`exItemEditSpin`), A spin editor is assigned to the current item. This value could be combined with one of the values from 0 to 7 or 256
- 1024 (`exItemEditVertical`), The editor is shown vertically rather than horizontally. This value has effect for `exItemEditSlider`, `exItemEditProgress` or `exItemEditScrollBar`
- `edit=<VALUE>`, specifies the caption to be shown in the item's edit field, where `<VALUE>` could be any string
- `mask=<VALUE>`, specifies the mask to be applied on a masked editor. This option is valid for `exItemEditMask` edit. Use the float option to allow masking floating point numbers. See [Masking](#) for more information about `<VALUE>` of the mask option. See [Masking Float](#) for more information about `<VALUE>` if the float option is used.
- `float=<VALUE>`, Specifies whether the mask field masks a floating point number. This option is valid for `exItemEditMask` edit. See [Masking Float](#) for more information about `<VALUE>` of mask option, if the float option is used. The `<VALUE>` could be **0** for standard masking field or **not zero** to specify that the field is masking a floating point.
- `border=<VALUE>`, specifies the border to be shown on the item's edit field, where `<VALUE>` could be one of the following:
 - **0** (`exEditBorderNone`), No border is shown.
 - **-1** (`exEditBorderInset`), shows an inset border
 - **1** (`exEditBorderSingle`), shows a frame border
- `editwidth=<VALUE>`, specifies the width to show the edit field inside the item, where `<VALUE>` could be a integer expression. A negative value indicates that the field goes to the end of the item
- `min=<VALUE>`, defines the minimum value of the edit field. The `<VALUE>` could be any integer expression, and specifies the minimum value for any slider, progress, scroll, spin, or range editor.
- `max=<VALUE>`, defines the maximum value of the edit field. The `<VALUE>` could be any integer expression, and specifies the maximum value for any slider, progress, scroll, spin, or range editor.
- `tick=<VALUE>`, defines where the ticks of the slider edit appear. This option is valid for `exItemEditSlider` edit. The `<VALUE>` could be one of the following values:
 - **0** (`exBottomRight`), The ticks are displayed on the bottom/right side.
 - **1** (`exTopLeft`), The ticks are displayed on the top/left side.
 - **2** (`exBoth`), The ticks are displayed on the both side.
 - **3** (`exNoTicks`), No ticks are displayed.
- `freq=<VALUE>`, indicates the ratio of ticks on the slider edit. This option is valid for `exItemEditSlider` edit. The `<VALUE>` could be a positive integer expression.
- `ticklabel=<VALUE>`, indicates the HTML label to be displayed on slider's ticks. This option is valid for `exItemEditSlider` edit. See [Tick Label Expression](#) for more information about `<VALUE>` of the ticklabel option.
- `small=<VALUE>`, indicates the amount by which the edit's position changes when the user presses the arrow key (left, right, or button). This option is valid for `exItemEditSlider`, `exItemEditScrollBar` edit. The `<VALUE>` could be a positive integer

expression.

- `large=<VALUE>`, indicates the amount by which the edit's position changes when the user presses the CTRL + arrow key (CTRL + left, CTRL + right). This option is valid for `exItemEditSlider`, `exItemEditScrollBar` edit. The `<VALUE>` could be a positive integer expression.
- `spin=<VALUE>`, specifies the step to advance when user clicks the editor's spin.. This option is valid for `exItemEditSpin` edit. The `<VALUE>` could be a positive integer expression.
- `ettp=<VALUE>`, specifies the HTML tooltip to be shown when the item's value is changed. This option is valid for `exItemEditSlider/exItemEditScrollBar` edit. The `<VALUE>` could be any string expression, including built-in HTML tags
- `arrow=<VALUE>`. The `<VALUE>` could be **0** for hiding the arrow or **not zero** to show the arrow. Indicates whether an item that has a sub-menu shows or hides its popup arrow. If the `<VALUE>` is missing, the item shows no arrow.
- `local=<VALUE>`. The `<VALUE>` could be **0** for standard popup or **not zero** to specify that the field is a local popup. Specifies whether the item's popup is shown as local. Clicking any item inside a local popup makes the popup itself to close including all its descendent sub-menus, without closing any ascendant sub-menus.
- `close=<VALUE>`, Specifies the way the hosting menu is closed when the user clicks the item. If the close flag is missing, the `<VALUE>` is 3 (`exCloseOnNonClickable`), by default. The `<VALUE>` could be one of the following values:
 - **0** (`exCloseOnClick`), The popup menu is closing when the user clicks the item.
 - **1** (`exCloseOnDbClick`), The popup menu is closing when the user double clicks the item.
 - **2** (`exCloseOnClickOutside`), The popup menu is closing when the user clicks outside of the menu.
 - **3** (`exCloseOnNonClickable`), The popup menu is closing when the user clicks a non-clickable item (regular items). The non-clickable items is any item that's not a separator, popup, disabled or check or radio items, clicking a check-box item will makes the check box to change its state instead closing the context menu.
- `popupapp=<VALUE>` indicates the visual appearance of the item's submenu when the popup is shown. The `<VALUE>` could be a predefined value like shown below, or an integer expression that refers an EBN object.
 - **0** (`NoBorder`)
 - **1** (`FlatBorder`)
 - **2** (`SunkenBorder`)
 - **3** (`RaisedBorder`)
 - **4** (`EtchedBorder`)
 - **5** (`BumpBorder`)
 - **6** (`ShadowBorder`)
 - **7** (`InsetBorder`)
 - **8** (`SingleBorder`)

- `itemsbg=<VALUE>`, specifies the items background color, where `<VALUE>` could be a RGB expression (`RGB(RR,GG,BB)`, where `RR` is the red value, the `GG` is the green value, and the `BB` is the blue value), or an integer expression to that refers an EBN object.
- `itemsbgshot=<VALUE>`, specifies the items background color, while the cursor hovers the items, where `<VALUE>` could be a RGB expression (`RGB(RR,GG,BB)`, where `RR` is the red value, the `GG` is the green value, and the `BB` is the blue value), or an integer expression to that refers an EBN object.
- `popupalign=<VALUE>`, Indicates how the item's sub-menu is aligned relative to the parent item. The `popupalign` has no effect for an item that displays a select- button. The `<VALUE>` could be a combination of one or more of the following values:
 - **0** (`exShowPopupAlignNone`), The popup menu is shown on top of the item, aligned to the left (no down and right, so up and left)
 - **1** (`exShowPopupAlignDown`), The popup menu is shown down. If missing, the popup menu is shown up.
 - **2** (`exShowPopupAlignRight`), The popup menu is shown aligned to the right, else if missing, the popup menu is shown aligned to the left.
- `popupat=<VALUE>`, specifies the identifier of the item where the current item's submenu/popup is displayed. The `<VALUE>` could be any integer expression. If there is no identifier with giving value, the option has no effect.
- `popupoffset=<VALUE>`, specifies the offset (horizontal,vertical) to display the item's submenu/popup relative to its default position.
- `itemspad=<VALUE>`, specifies the padding (space between the menu border and the item content) to display the items. The `<VALUE>` is a list of coordinates such as `left,top,right,bottom`
- `visible=<VALUE>`, specifies the maximum number of visible items at one time, where the `<VALUE>` could be any integer expression.
- `tab=<VALUE>`, specifies the identifier of the item/tab where the current group-popup is shown instead. The `<VALUE>` could be any integer expression. If there is no identifier with giving value, the option has no effect.
- `itemsbgext=<VALUE>`, indicates additional colors, text, images that can be displayed on the items background using the [EBN String Format](#). The `<VALUE>` should be in [EBN String Format](#). For instance, `[itemsbgext=bottom[2],bottom[16,text=`</fgcolor><fgcolor 6D6AAA>Views</fgcolor><fgcolor A0A0A0>`,align=0x21]]`, shows the Views aligned to the bottom, with a different foreground color.

Masking, (mask option)

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

!(999) 000 0000;1;;select=1,empty,overtypewarning=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 *overtyp*e 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;overtyp,e,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)
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,overtime", 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.
- **overtime**, 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;overtypetype,inserttype", indicates that the field enter in over-type mode, and insert-type mode is allowed. The "###:###;0;overtypetype", 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.
- **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,overtypetype", 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,overtyp=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

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.

Masking-Float, (mask, float option)

The `[mask=<VALUE>]` property may indicate the followings, if the `[float=-1]` is present

- **negative number**: if the first character in the mask is - (minus) the control supports negative numbers. Pressing the - key will toggle the sign of the number. The + sign is never displayed.
- **decimal symbol**: the last character that's different than # (digit), or 0 (zero) indicates the decimal symbol. If it is not present the control mask a floating point number without decimals.
- **thousand symbol**: the thousand symbol is the last character that's not a # (digit), 0 (zero) or it is not the decimal symbol as explained earlier, if present.
- the maximum **number of decimals** in the number (the # or 0 character after the decimal symbol)
- the maximum number of digits in the integer part (the number of # or 0 character

before decimal symbol)

- the **0** character indicates a **leading-zero**. The count of 0 (zero) characters before decimal character indicates the leading-zero for integer part of the control, while the count of 0 (zero) characters after the decimal separator indicates the leading-zero for decimal part of the control. For instance, the Mask on "-###,###,##0.00", while the control's Text property is 1, the control displays 1.00, if 1.1 if displays 1.10, and if empty, the 0.00 is displayed.

If the <VALUE> property is empty, the control takes the settings for the regional options like: Decimal Symbol , No. of digits after decimal, Digit grouping symbol.

Here are few samples:

The <VALUE>"-###.###.##0,00" filter floating point numbers a number for German settings ("," is the decimal sign, "." is the thousands separator). This format displays leading-zeros.

The <VALUE>"-###.###.###,##" filter floating point numbers a number for German settings ("," is the decimal sign, "." is the thousands separator)

The <VALUE>"-###,###,###.##" filter floating point numbers a number for English settings ("." is the decimal sign, "," is the thousands separator)

The <VALUE>"####" indicates a max-4 digit number (positive) without a decimal symbol and without digit grouping

The <VALUE>"-##.##" filters a floating point number from the -99.9 to 99.9 ("." is the decimal sign, no thousands separator)

The <VALUE>"#,###.##" filters a floating point number from the 0 to 9,999.99 with digit grouping ("." is the decimal sign, "," is the thousands separator).

Tick Label Expression, (ticklabel option)



For instance:

- "value", shows the values for each tick.
- " (value=current ? '<fgcolor=FF0000>' : ") + value", shows the current slider's position with a different color and font.
- "value = current ? value : """, shows the value for the current tick only.
- "(value = current ? '' : ") + (value array 'ab bc cd de ef fg gh hi ij jk kl' split ' ')" displays different captions for slider's values.

The The <VALUE> of [ticklabel] option is a formatted expression which result may include the [HTML](#) tags.

The The <VALUE> of [ticklabel] option indicates a formatting expression that may use the following predefined keywords:

- **value** gets the slider's position to be displayed
- **current** gets the current slider's value.
- **vmin** gets the slider's minimum value.
- **vmax** gets the slider's maximum value.
- **smin** gets the slider's selection minimum value.
- **smax** gets the slider's selection maximum value.

The supported binary arithmetic operators are:

- * (multiplicity operator), priority 5
- / (divide operator), priority 5
- **mod** (remainder operator), priority 5
- + (addition operator), priority 4 (concatenates two strings, if one of the operands is of string type)
- - (subtraction operator), priority 4

The supported unary boolean operators are:

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

The supported binary boolean operators are:

- **or** (or operator), priority 2
- **and** (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 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 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 than iif (immediate if operator) alternative.

- **case()** (*case operator*) returns and executes one of n expressions, depending on the evaluation of the expression (IIF - immediate IF operator is a binary case() operator). The syntax for case() operator is:

"expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)"

If the default part is missing, the case() operator returns the value of the expression if it is not found in the collection of cases (c1, c2, ...). For instance, if the value of expression is not any of c1, c2, the default_expression is executed and returned. If the value of the expression is c1, then the case() operator executes and returns the expression1. The default, c1, c2, c3, ... must be constant elements as numbers, dates or strings. For instance, the "date(shortdate(value)) case (default:0 ; #1/1/2002#:1 ; #2/1/2002#:1; #4/1/2002#:1; #5/1/2002#:1)" indicates that only #1/1/2002#, #2/1/2002#, #4/1/2002# and #5/1/2002# dates returns 1, since the others returns 0. For instance the following sample specifies the hour being non-working for specified dates: "date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)" statement indicates the working hours for dates as follows:

- - #4/1/2009#, from hours 06:00 AM to 12:00 PM
 - #4/5/2009#, from hours 07:00 AM to 10:00 AM and hours 03:00PM, 04:00PM, 06:00PM and 10:00PM
 - #5/1/2009#, from hours 12:00 AM to 08:00 AM

The *in*, *switch* and *case()* use binary search to look for elements so they are faster than using 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. For instance type(%0) = 8 specifies the cells that contains string values.

Here's few predefined types:

- 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
- **str** (unary operator) converts the expression to a string
- **dbl** (unary operator) converts the expression to a number
- **date** (unary operator) converts the expression to a date, based on your regional settings
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS.

Other known operators for numbers are:

- **int** (unary operator) retrieves the integer part of the number
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2
- 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

Other known operators for strings are:

- **len** (unary operator) retrieves the number of characters in the string
- **lower** (unary operator) returns a string expression in lowercase letters
- **upper** (unary operator) returns a string expression in uppercase letters
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names
- **ltrim** (unary operator) removes spaces on the left side of a string
- **rtrim** (unary operator) removes spaces on the right side of a string

- **trim** (unary operator) removes spaces on both sides of a string
- **startswith** (binary operator) specifies whether a string starts with specified string
- **endwith** (binary operator) specifies whether a string ends with specified string
- **contains** (binary operator) specifies whether a string contains another specified string
- **left** (binary operator) retrieves the left part of the string
- **right** (binary operator) retrieves the right part of the string
- a **mid** b (binary operator) retrieves the middle part of the string a starting from b (1 means first position, and so on)
- a **count** b (binary operator) retrieves the number of occurrences of the b in a
- a **replace** b **with** c (double binary operator) replaces in a the b with c, and gets the result.
- a **split** b, 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

Other known operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel.
- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance the timeF(1:23 PM) returns "13:23:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel.
- **shortdateF** (unary operator) formats a date as a date string using the "MM/DD/YYYY" format. For instance the shortdateF(December 31, 1971 11:00 AM) returns "12/31/1971".
- **dateF** (unary operator) converts the date expression to a string expression in "MM/DD/YYYY HH:MM:SS" format.
- **longdate** (unary operator) formats a date as a date string using the long date format, as specified in the control's panel.
- **year** (unary operator) retrieves the year of the date (100,...,9999)
- **month** (unary operator) retrieves the month of the date (1, 2,...,12)
- **day** (unary operator) retrieves the day of the date (1, 2,...,31)
- **yearday** (unary operator) retrieves the number of the day in the year, or the days since January 1st (0, 1,...,365)
- **weekday** (unary operator) retrieves the number of days since Sunday (0 - Sunday, 1 - Monday,..., 6 - Saturday)
- **hour** (unary operator) retrieves the hour of the date (0, 1, ..., 23)
- **min** (unary operator) retrieves the minute of the date (0, 1, ..., 59)
- **sec** (unary operator) retrieves the second of the date (0, 1, ..., 59)

The The <VALUE> of [ticklabel] option can display labels using the following built-in HTML

tags:

- **** displays the text in **bold**.
- **<i></i>** displays the text in *italics*.
- **<u></u>** underlines the text.
- **<s></s>** Strike-through text
- **** displays portions of text with a different font and/or different size. For instance, the **bit** draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, **bit** displays the bit text using the current font, but with a different size.
- **<fgcolor=RRGGBB></fgcolor>** displays text with a specified **foreground** color. The RR, GG or BB should be hexa values and indicates red, green and blue values.
- **<bcolor=RRGGBB></bcolor>** displays text with a specified **background** color. The RR, GG or BB should be hexa values and indicates red, green and blue values.
- **
** a forced line-break
- **<solidline>** The next line shows a solid-line on top/bottom side. If has no effect for a single line caption.
- **<dotline>** The next line shows a dot-line on top/bottom side. If has no effect for a single line caption.
- **<upline>** The next line shows a solid/dot-line on top side. If has no effect for a single line caption.
- **<r>** Right aligns the text
- **<c>** Centers the text
- **number[:width]** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **key[:width]** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&**; (&), **<**; (<), **>**; (>), **&qout;** (") and **&#number** (the character with specified code), For instance, the **€** displays the EUR character, in UNICODE configuration. The **&** ampersand is only recognized as markup when it is followed by a known letter or a # character and a digit. For instance if you want to display **bold** in HTML caption you can use **bold**;

EBN String Format, (itemsbgext option)

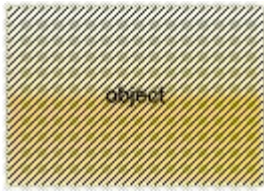
The **EBN String Format** syntax in BNF notation is defined like follows:

```
<EBN> ::= <elements> | <root> "(" [<elements> "]" ")"
<elements> ::= <element> [ "," <elements> ]
<root> ::= "root" [ <attributes> ] | [ <attributes> ]
<element> ::= <anchor> [ <attributes> ] [ "(" [<elements> "]" ")" ]
<anchor> ::= "none" | "left" | "right" | "client" | "top" | "bottom"
<attributes> ::= "[" [<client> "," <attribute> [ "," <attributes> ] "]"
<client> ::= <expression> | <expression> "," <expression> "," <expression> ","
<expression>
<expression> ::= <number> | <number> "%"
<attribute> ::= <backcolor> | <text> | <wordwrap> | <align> | <pattern> |
<patterncolor> | <frame> | <framethick> | <data> | <others>
<equal> ::= "="
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<decimal> ::= <digit> <decimal>
<hexadigit> ::= <digit> | "A" | "B" | "C" | "D" | "E" | "F"
<hexa> ::= <hexadigit> <hexa>
<number> ::= <decimal> | "0x" <hexa>
<color> ::= <rgbcolor> | number
<rgbcolor> ::= "RGB" "(" <number> "," <number> "," <number> ")"
<string> ::= "\"" <characters> "\"" | "\"" <characters> "\"" | "<characters> "
<characters> ::= <char> | <characters>
<char> ::= <any_character_excepts_null>
<backcolor> ::= "back" <equal> <color>
<text> ::= "text" <equal> <string>
<align> ::= "align" <equal> <number>
<pattern> ::= "pattern" <equal> <number>
<patterncolor> ::= "patterncolor" <equal> <color>
<frame> ::= "frame" <equal> <color>
<data> ::= "data" <equal> <number> | <string>
<framethick> ::= "framethick"
<wordwrap> ::= "wordwrap"
```

Others like: pic, stretch, hstretch, vstretch, transparent, from, to are reserved for future use only.

Here's a few easy samples:

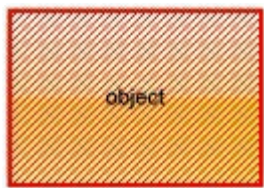
- "[pattern=6]", shows the BDiagonal pattern on the object's background.



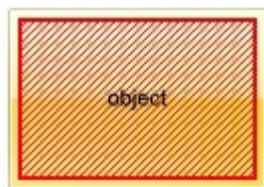
- "[frame=RGB(255,0,0),framethick]", draws a red thick-border around the object.



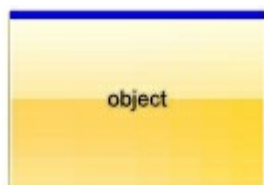
- "[frame=RGB(255,0,0),framethick,pattern=6,patterncolor=RGB(255,0,0)]", draws a red thick-border around the object, with a patter inside.



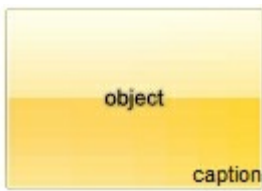
- "[[patterncolor=RGB(255,0,0)]
(none[(4,4,100%-8,100%-8),pattern=0x006,patterncolor=RGB(255,0,0),frame=RGB(255,0,0),framethick])]", draws a red thick-border around the object, with a patter inside, with a 4-pixels wide padding:



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



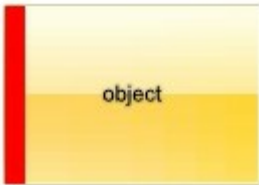
- "[text=`caption`,align=0x22]", shows the caption string aligned to the bottom-right side of the object's background.



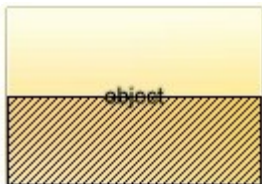
- "[text=`flag`,align=0x11]" shows the flag picture and the sweden string aligned to the bottom side of the object.



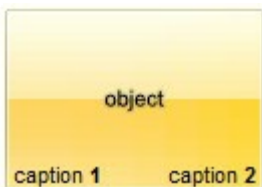
- "left[10,back=RGB(255,0,0)]", draws a red line on the left side of the object's background, of 10-pixels wide.



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



- "root[text=`caption 2` ,align=0x22](client[text=`caption 1` ,align=0x20])", shows the caption **1** aligned to the bottom-left side, and the caption **2** to the bottom-right side

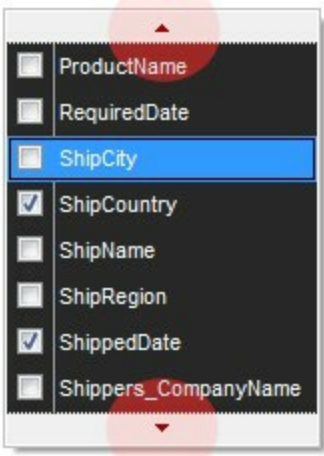


property Items.VisibleItemCount as Long

Specifies the maximum number of visible items at one time.

Type	Description
Long	A long expression that indicates the maximum number of visible items into the popup menu.

By default, the VisibleItemCount property is 12. The control adds scroll buttons to a popup menu, if the menu contains more items than VisibleItemCount property. Use the [Visible](#) property to specify whether an item is visible or hidden. Use the [Add](#) method to add new items to the control. The VisibleItemCount property specifies the number of items being visible without scroll option.



OleEvent object

The OleEvent object holds information about an event fired by an ActiveX control hosted by in item that was created using the [Add\(,SubControl\)](#) method.

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

property OleEvent.CountParam as Long

Retrieves the count of the OLE event's arguments.

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

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.

property OleEvent.ID as Long

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

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

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

property OleEvent.Name as String

Retrieves the original name of the fired event.

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

Use the 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.

property OleEvent.Param (Item as Variant) as OleEventParam

Retrieves an OleEventParam object given either the index of the parameter, or its name.

Type	Description
Item as Variant	A long expression that indicates the argument's index or a string expression that indicates the argument's name.
OleEventParam	An OleEventParam object that holds information about a parameter of an event.

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.

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.

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.

Ribbon 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: {DDF58CFA-750F-45E0-8A00-CFBE431702E2}. The object's program identifier is: "Exontrol.Ribbon". The /COM object module is: "ExRibbon.dll"

The eXRibbon component, similar with the Microsoft Office's "Fluent User Interface", or Ribbon, allows you to display graphical control elements in the form of a set of toolbars placed on several tabs. The eXRibbon component is written from scratch, uses [EBN](#) technology to let the user changes its visual appearance using skins, and requires no dependencies to Microsoft Office, Microsoft Ribbon API, or any other third party library. The major difference between our ribbon implementation and others, is that you can display and use built-editors (like sliders, spin, progress, ...) anywhere on the control. The control supports the following 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.
AllowToggleRadio	Allows or prevents toggling the radio state.
AllowToolTip	Allows or prevents showing the item's tooltip.
Appearance	Retrieves or sets the popup's appearance.
AttachTemplate	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
BackColor	Specifies the control's background color.
Background	Returns or sets a value that indicates the background color for parts in the control.
Cursor	Gets or sets the cursor that is displayed when the mouse pointer hovers the control.
Debug	Retrieves or sets a value that indicating whether the item's identifier is visible.
Enabled	Enables or disables the control.
EventParam	Retrieves or sets a value that indicates the current's event parameter.
ExecuteTemplate	Executes a template and returns the result.
Font	Retrieves or sets the control's font.
ForeColor	Specifies the control's foreground color.
FormatABC	Formats the A,B,C values based on the giving expression and returns the result.

Get	Retrieves an array of Item objects that meet the criteria.
GetChecked	Retrieves an array of Item objects, that displays a check box which is checked.
GetRadio	Retrieves an array of Item objects of radio type in the same group, that are checked.
GetUnchecked	Retrieves an array of Item objects, that displays a check box which is unchecked.
HTMLPicture	Adds or replaces a picture in HTML captions.
hWnd	Retrieves the control's window handle.
Images	Sets at runtime the control's image list. The Handle should be a handle to an Images List Control.
ImageSize	Specifies the size to show the icons, check-boxes, radio-buttons, drop-down arrows and so on.
Item	Returns a specific Item object giving its identifier or caption.
ItemFromPoint	Retrieves the item from the cursor.
Items	Retrieves the control's Items collection.
LocalAppearance	Retrieves or sets the local popup's appearance.
Locked	Locks or unlocks the control.
Notifier	Retrieves or sets the handle of the window that receives notifications/WM_COMMAND messages.
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
PopupAppearance	Retrieves or sets the popup's appearance.
PopupFlatAppearance	Specifies whether the control shows a flat appearance for sub-menus.
PopupFlatBackColor	Specifies the background color to show the left part of the flat sub-menus.
PopupFlatImageWidth	Specifies the width of the flat portion of the sub-menu, in pixels.
PopupIncrementalSearch	Specifies how the popup searches for the objects while user types characters.
PopupVisibility	Specify the popup's visibility in percents: 90% is barely visible, and 10% is nearly opaque.

Refresh	Refreshes the control.
Replacelcon	Adds a new icon, replaces an icon or clears the control's image list.
RequiredHeight	Indicates the height to fit all elements within the control.
RequiredWidth	Indicates the width to fit all elements within the control.
SelBackColor	Retrieves or sets a value that indicates the selection background color.
SelForeColor	Retrieves or sets a value that indicates the selection foreground color.
ShortcutKeyAlignH	Defines the alignment of the UI shortcut keys relative to the item that displays it, when the holder arranges the items horizontally.
ShortcutKeyAlignV	Defines the alignment of the UI shortcut keys relative to the item that displays it, when the holder arranges the items vertically.
ShortcutKeyExtPaddingH	Specifies the padding to be applied to the item (arranged horizontally) that displays the UI shortcut, so the ShortcutKeyAlign property aligns the UI shortcut relative to this.
ShortcutKeyExtPaddingV	Specifies the padding to be applied to the item (arranged vertically) that displays the UI shortcut, so the ShortcutKeyAlign property aligns the UI shortcut relative to this.
ShortcutKeyFormat	Specifies the expression that determines the format (including HTML tags) to display the shortcut keys.
ShortcutKeyPadding	Specifies the UI shortcut key's content padding.
ShortcutKeyPressedModifiers	Indicates the combination of modifier keys whose shortcut keys are currently visible.
ShortcutKeysInfo	Returns the list of shortcut keys that are currently available.
ShortcutKeyTransparent	Indicates the percent of transparency the shortcut keys are being displayed with.
ShortcutKeyVisible	Gets or sets a value that specifies whether the control's shortcut keys are visible or hidden.
ShowCheckedAsSelected	Specifies whether the checked items shows as selected.
ShowImageList	Specifies whether the control's image list window is visible

or hidden.

[ShowPopupArrow](#)

Indicates the type of the arrow to be shown when the item displays the sub-menu.

[ShowPopupEffect](#)

Specifies the effect to show the popup menu when clicking an item, such as scrolling, lighting up, and so on.

[ShowToolTip](#)

Shows the specified tooltip at given position.

[Template](#)

Specifies the control's template.

[TemplateDef](#)

Defines inside variables for the next Template/ExecuteTemplate call.

[TemplatePut](#)

Defines inside variables for the next Template/ExecuteTemplate call.

[ToolTipDelay](#)

Specifies the time in ms that passes before the ToolTip appears.

[ToolTipFont](#)

Retrieves or sets the tooltip's font.

[ToolTipPopDelay](#)

Specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control.

[ToolTipWidth](#)

Specifies a value that indicates the width of the tooltip window, in pixels.

[ToString](#)

Loads or saves the Items collection using string representation (shortcut of Items.ToString property).

[ToTemplate](#)

Generates the control's template.

[Update](#)

Updates the control.

[UseVisualTheme](#)

Specifies whether the control uses the current visual theme to display certain UI parts.

[Version](#)

Retrieves the control's version.

[VisualAppearance](#)

Retrieves the control's appearance.

property Ribbon.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 specifies 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, which indicates that you can copy the ribbon's content in x-script format by pressing the Shift + Ctrl + Alt + Insert keys combination. If the function succeeded the generated x-script code is copied to your clipboard and a beep is played. In order to execute the generated code, you can use the [eXHelper](#) tool. Based on x-script, the exHelper tool can generate code in different programming languages such as C++, VB, Delphi, and much more. The [Template / ExecuteTemplate](#) can be used to execute the x-script being generated. The [ToTemplate](#) property generates x-script from the control's content.

property Ribbon.AllowToggleRadio as Boolean

Allows or prevents toggling the radio state.

Type	Description
Boolean	A Boolean expression that specifies whether the radio-buttons allow toggling its value.

By default, the AllowToggleRadio property is False. The AllowToggleRadio property on True, allows a radio button to set on zero (unchecked), if the user clicks twice the radio button. Usually, clicking a radio-button makes the previously checked radio-button in the same group, to be un-checked, and the newly clicked item to be checked. Now, if the AllowToggleRadio property is True, clicking again the radio-button, allows the radio-button to be un-checked, so allows a radio group to have no radio button checked. The control fires the [CheckItem](#) event once a radio-button is clicked. The [Radio](#) property specifies whether the item displays a radio-button. The [RadioGroup](#) property specifies a group of radio-buttons. A radio group allows a single radio-item to be checked. The [Checked](#) property specifies whether the item is checked or un-checked. The [GetRadio](#) method gets a safe array with the radio-items being checked within a radio group. Use the [Background\(exRadioButtonState0\)/Background\(exRadioButtonState1\)](#) property to specify the visual appearance of the radio-buttons in the control. Use the [UseVisualTheme](#) property to specify whether the visual appearance for the radio-buttons to be as indicated by the current XP theme.

I am using radio-buttons, the question is it possible to uncheck the radio-buttons, so no button is pressed in the group?

VBA (MS Access, Excell...)

```
With Ribbon1
    .AllowToggleRadio = True
With .Items
    With .Add("Radio 1",0,1000)
        .Radio = True
        .RadioGroup = 100
    End With
    With .Add("Radio 2",0,1001)
        .Radio = True
        .RadioGroup = 100
    End With
    With .Add("Radio 2",0,1003)
```



```
.Radio = True
.RadioGroup = 100
End With
End With
.Refresh
End With
```

VB6

```
With Ribbon1
.AllowToggleRadio = True
With .Items
With .Add("Radio 1",0,1000)
.Radio = True
.RadioGroup = 100
End With
With .Add("Radio 2",0,1001)
.Radio = True
.RadioGroup = 100
End With
With .Add("Radio 2",0,1003)
.Radio = True
.RadioGroup = 100
End With
End With
.Refresh
End With
```

VB.NET

```
With Exribbon1
.AllowToggleRadio = True
With .Items
With .Add("Radio 1",0,1000)
.Radio = True
.RadioGroup = 100
End With
With .Add("Radio 2",0,1001)
```

```

        .Radio = True
        .RadioGroup = 100
    End With
    With .Add("Radio 2",0,1003)
        .Radio = True
        .RadioGroup = 100
    End With
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    .AllowToggleRadio = True
    With .Items
        With .Add("Radio 1",0,1000)
            .Radio = True
            .RadioGroup = 100
        End With
        With .Add("Radio 2",0,1001)
            .Radio = True
            .RadioGroup = 100
        End With
        With .Add("Radio 2",0,1003)
            .Radio = True
            .RadioGroup = 100
        End With
    End With
    .Refresh()
End With

```

C++

```

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

```

```

#import <ExRibbon.dll>
using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
spRibbon1->PutAllowToggleRadio(VARIANT_TRUE);
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Radio 1",long(0),long(1000));
    var_Item->PutRadio(VARIANT_TRUE);
    var_Item->PutRadioGroup(100);
    EXRIBBONLib::IItemPtr var_Item1 = var_Items->Add(L"Radio
2",long(0),long(1001));
    var_Item1->PutRadio(VARIANT_TRUE);
    var_Item1->PutRadioGroup(100);
    EXRIBBONLib::IItemPtr var_Item2 = var_Items->Add(L"Radio
2",long(0),long(1003));
    var_Item2->PutRadio(VARIANT_TRUE);
    var_Item2->PutRadioGroup(100);
spRibbon1->Refresh();

```

C++ Builder

```

Ribbon1->AllowToggleRadio = true;
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    Exribbonlib_tlb::IItemPtr var_Item = var_Items->Add(L"Radio
1",TVariant(0),TVariant(1000));
    var_Item->Radio = true;
    var_Item->RadioGroup = 100;
    Exribbonlib_tlb::IItemPtr var_Item1 = var_Items->Add(L"Radio
2",TVariant(0),TVariant(1001));
    var_Item1->Radio = true;
    var_Item1->RadioGroup = 100;
    Exribbonlib_tlb::IItemPtr var_Item2 = var_Items->Add(L"Radio
2",TVariant(0),TVariant(1003));
    var_Item2->Radio = true;

```

```
var_Item2->RadioGroup = 100;  
Ribbon1->Refresh();
```

C#

```
exribbon1.AllowToggleRadio = true;  
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Radio 1",0,1000);  
var_Item.Radio = true;  
var_Item.RadioGroup = 100;  
exontrol.EXRIBBONLib.Item var_Item1 = var_Items.Add("Radio 2",0,1001);  
var_Item1.Radio = true;  
var_Item1.RadioGroup = 100;  
exontrol.EXRIBBONLib.Item var_Item2 = var_Items.Add("Radio 2",0,1003);  
var_Item2.Radio = true;  
var_Item2.RadioGroup = 100;  
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
Ribbon1.AllowToggleRadio = true;  
var var_Items = Ribbon1.Items;  
var var_Item = var_Items.Add("Radio 1",0,1000);  
var_Item.Radio = true;  
var_Item.RadioGroup = 100;  
var var_Item1 = var_Items.Add("Radio 2",0,1001);  
var_Item1.Radio = true;  
var_Item1.RadioGroup = 100;  
var var_Item2 = var_Items.Add("Radio 2",0,1003);
```

```
var_Item2.Radio = true;
var_Item2.RadioGroup = 100;
Ribbon1.Refresh();
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Ribbon1
    .AllowToggleRadio = True
  With .Items
    With .Add("Radio 1",0,1000)
      .Radio = True
      .RadioGroup = 100
    End With
    With .Add("Radio 2",0,1001)
      .Radio = True
      .RadioGroup = 100
    End With
    With .Add("Radio 2",0,1003)
      .Radio = True
      .RadioGroup = 100
    End With
  End With
  .Refresh
End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
axRibbon1.AllowToggleRadio = true;
EXRIBBONLib.Items var_Items = axRibbon1.Items;
    EXRIBBONLib.Item var_Item = var_Items.Add("Radio 1",0,1000);
        var_Item.Radio = true;
        var_Item.RadioGroup = 100;
    EXRIBBONLib.Item var_Item1 = var_Items.Add("Radio 2",0,1001);
        var_Item1.Radio = true;
        var_Item1.RadioGroup = 100;
    EXRIBBONLib.Item var_Item2 = var_Items.Add("Radio 2",0,1003);
        var_Item2.Radio = true;
        var_Item2.RadioGroup = 100;
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Item,com_Item1,com_Item2,com_Items;
    anytype var_Item,var_Item1,var_Item2,var_Items;
    ;

    super();

    exribbon1.AllowToggleRadio(true);
    var_Items = exribbon1.Items(); com_Items = var_Items;
        var_Item = com_Items.Add("Radio
1",COMVariant::createFromInt(0),COMVariant::createFromInt(1000)); com_Item =
var_Item;
            com_Item.Radio(true);
            com_Item.RadioGroup(100);
        var_Item1 = com_Items.Add("Radio
2",COMVariant::createFromInt(0),COMVariant::createFromInt(1001)); com_Item1 =
var_Item1;
```

```

        com_Item1.Radio(true);
        com_Item1.RadioGroup(100);
        var_Item2 = com_Items.Add("Radio
2",COMVariant::createFromInt(0),COMVariant::createFromInt(1003)); com_Item2 =
var_Item2;
        com_Item2.Radio(true);
        com_Item2.RadioGroup(100);
        exribbon1.Refresh();
    }

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    AllowToggleRadio := True;
    with Items do
    begin
        with Add('Radio 1',TObject(0),TObject(1000)) do
        begin
            Radio := True;
            RadioGroup := 100;
        end;
        with Add('Radio 2',TObject(0),TObject(1001)) do
        begin
            Radio := True;
            RadioGroup := 100;
        end;
        with Add('Radio 2',TObject(0),TObject(1003)) do
        begin
            Radio := True;
            RadioGroup := 100;
        end;
    end;
    Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  AllowToggleRadio := True;
  with Items do
  begin
    with Add('Radio 1',OleVariant(0),OleVariant(1000)) do
    begin
      Radio := True;
      RadioGroup := 100;
    end;
    with Add('Radio 2',OleVariant(0),OleVariant(1001)) do
    begin
      Radio := True;
      RadioGroup := 100;
    end;
    with Add('Radio 2',OleVariant(0),OleVariant(1003)) do
    begin
      Radio := True;
      RadioGroup := 100;
    end;
  end;
  Refresh();
end

```

VFP

```

with thisform.Ribbon1
.AllowToggleRadio = .T.
with .Items
  with .Add("Radio 1",0,1000)
    .Radio = .T.
    .RadioGroup = 100
  endwith
  with .Add("Radio 2",0,1001)
    .Radio = .T.
    .RadioGroup = 100
  endwith

```



```
with .Add("Radio 2",0,1003)
    .Radio = .T.
    .RadioGroup = 100
endwith
endwith
.Refresh
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Item1,var_Item2,var_Items

oRibbon = form.Activex1.nativeObject
oRibbon.AllowToggleRadio = true
var_Items = oRibbon.Items
    var_Item = var_Items.Add("Radio 1",0,1000)
        var_Item.Radio = true
        var_Item.RadioGroup = 100
    var_Item1 = var_Items.Add("Radio 2",0,1001)
        var_Item1.Radio = true
        var_Item1.RadioGroup = 100
    var_Item2 = var_Items.Add("Radio 2",0,1003)
        var_Item2.Radio = true
        var_Item2.RadioGroup = 100
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Item2 as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
oRibbon.AllowToggleRadio = .t.
var_Items =oRibbon.Items
```

```

var_Item = var_Items.Add("Radio 1",0,1000)
    var_Item.Radio = .t.
    var_Item.RadioGroup = 100
var_Item1 = var_Items.Add("Radio 2",0,1001)
    var_Item1.Radio = .t.
    var_Item1.RadioGroup = 100
var_Item2 = var_Items.Add("Radio 2",0,1003)
    var_Item2.Radio = .t.
    var_Item2.RadioGroup = 100
oRibbon.Refresh()

```

Visual Objects

```

local var_Item,var_Item1,var_Item2 as Item
local var_Items as Items

oDCOCX_Exontrol1:AllowToggleRadio := true
var_Items := oDCOCX_Exontrol1:Items
    var_Item := var_Items.Add("Radio 1",0,1000)
        var_Item:Radio := true
        var_Item:RadioGroup := 100
    var_Item1 := var_Items.Add("Radio 2",0,1001)
        var_Item1:Radio := true
        var_Item1:RadioGroup := 100
    var_Item2 := var_Items.Add("Radio 2",0,1003)
        var_Item2:Radio := true
        var_Item2:RadioGroup := 100
oDCOCX_Exontrol1.Refresh()

```

PowerBuilder

```

OleObject oRibbon,var_Item,var_Item1,var_Item2,var_Items

oRibbon = ole_1.Object
oRibbon.AllowToggleRadio = true
var_Items =oRibbon.Items

```

```
var_Item = var_Items.Add("Radio 1",0,1000)
    var_Item.Radio = true
    var_Item.RadioGroup = 100
var_Item1 = var_Items.Add("Radio 2",0,1001)
    var_Item1.Radio = true
    var_Item1.RadioGroup = 100
var_Item2 = var_Items.Add("Radio 2",0,1003)
    var_Item2.Radio = true
    var_Item2.RadioGroup = 100
oRibbon.Refresh()
```

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Set **ComAllowToggleRadio** to True

Variant voltems

Get Comltems to voltems

Handle holtems

Get Create (RefClass(cComltems)) to holtems

Set pvComObject of holtems to voltems

Variant voltem

Get ComAdd of holtems "Radio 1" 0 1000 to voltem

Handle holtem

Get Create (RefClass(cComltem)) to holtem

Set pvComObject of holtem to voltem

Set ComRadio of holtem to True

Set ComRadioGroup of holtem to 100

Send Destroy to holtem

Variant voltem1

Get ComAdd of holtems "Radio 2" 0 1001 to voltem1

Handle holtem1

Get Create (RefClass(cComltem)) to holtem1

Set pvComObject of holtem1 to voltem1

Set ComRadio of holtem1 to True

Set ComRadioGroup of holtem1 to 100

```

Send Destroy to holtem1
Variant voltem2
Get ComAdd of holtems "Radio 2" 0 1003 to voltem2
Handle holtem2
Get Create (RefClass(cComItem)) to holtem2
Set pvComObject of holtem2 to voltem2
    Set ComRadio of holtem2 to True
    Set ComRadioGroup of holtem2 to 100
Send Destroy to holtem2
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem, oltem1, oltem2
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oRibbon:AllowToggleRadio := .T.
    oltems := oRibbon:Items()

```

```
oltem := oltems:Add("Radio 1",0,1000)
  oltem:Radio := .T.
  oltem:RadioGroup := 100
oltem1 := oltems:Add("Radio 2",0,1001)
  oltem1:Radio := .T.
  oltem1:RadioGroup := 100
oltem2 := oltems:Add("Radio 2",0,1003)
  oltem2:Radio := .T.
  oltem2:RadioGroup := 100
oRibbon:Refresh()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
  nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
  oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Ribbon.AllowToolTip as Boolean

Allows or prevents showing the item's tooltip.

Type	Description
Boolean	A Boolean expression that specifies whether the control displays the item's tooltip when the cursor hovers the item.

By default, the AllowToolTip property is True. Use the AllowToolTip property on False, to prevent shown the item's tooltip when the cursor hovers the item. The [Tooltip](#) property assigns a HTML tooltip to an item, that's displayed only when the cursor hovers the item. The [TooltipTitle](#) property specifies the title for the item's tooltip. The [TooltipDelay](#) property specifies the time until the tooltip is shown. Use the [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipWidth](#) property to specify the width of the tooltip window. The [ToolTipFont](#) property specifies 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.

How do I disable showing the tooltip for all control?

VBA (MS Access, Excell...)

```
With Ribbon1
    .AllowToolTip = False
    With .Items
        .Add("Item").Tooltip = "this is a bit of text to be shown when cursor hovers it"
        .Add("Item").Tooltip = "this is a bit of text to be shown when cursor hovers it"
    End With
    .Refresh
End With
```

VB6

```
With Ribbon1
    .AllowToolTip = False
    With .Items
        .Add("Item").Tooltip = "this is a bit of text to be shown when cursor hovers it"
        .Add("Item").Tooltip = "this is a bit of text to be shown when cursor hovers it"
    End With
```

```
.Refresh  
End With
```

VB.NET

```
With Exribbon1  
    .AllowToolTip = False  
    With .Items  
        .Add("Item").Tooltip = "this is a bit of text to be shown when cursor hovers it"  
        .Add("Item").Tooltip = "this is a bit of text to be shown when cursor hovers it"  
    End With  
    .Refresh()  
End With
```

VB.NET for /COM

```
With AxRibbon1  
    .AllowToolTip = False  
    With .Items  
        .Add("Item").Tooltip = "this is a bit of text to be shown when cursor hovers it"  
        .Add("Item").Tooltip = "this is a bit of text to be shown when cursor hovers it"  
    End With  
    .Refresh()  
End With
```

C++

```
/*  
    Copy and paste the following directives to your header file as  
    it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control  
    Library'  
  
    #import <ExRibbon.dll>  
    using namespace EXRIBBONLib;  
*/  
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-  
>GetControlUnknown();  
spRibbon1->PutAllowToolTip(VARIANT_FALSE);
```

```
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
var_Items->Add(L"Item",vtMissing,vtMissing)->PutTooltip(L"this is a bit of text to
be shown when cursor hovers it");
var_Items->Add(L"Item",vtMissing,vtMissing)->PutTooltip(L"this is a bit of text to
be shown when cursor hovers it");
spRibbon1->Refresh();
```

C++ Builder

```
Ribbon1->AllowToolTip = false;
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
var_Items->Add(L"Item",TNoParam(),TNoParam())->Tooltip = L"this is a bit of text
to be shown when cursor hovers it";
var_Items->Add(L"Item",TNoParam(),TNoParam())->Tooltip = L"this is a bit of text
to be shown when cursor hovers it";
Ribbon1->Refresh();
```

C#

```
exribbon1.AllowToolTip = false;
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
var_Items.Add("Item",null,null).Tooltip = "this is a bit of text to be shown when
cursor hovers it";
var_Items.Add("Item",null,null).Tooltip = "this is a bit of text to be shown when
cursor hovers it";
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
```



```

{
    Ribbon1.AllowToolTip = false;
    var var_Items = Ribbon1.Items;
    var_Items.Add("Item",null,null).Tooltip = "this is a bit of text to be shown when
cursor hovers it";
    var_Items.Add("Item",null,null).Tooltip = "this is a bit of text to be shown when
cursor hovers it";
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload= 'Init()' >
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        .AllowToolTip = False
        With .Items
            .Add("Item").Tooltip = "this is a bit of text to be shown when cursor hovers it"
            .Add("Item").Tooltip = "this is a bit of text to be shown when cursor hovers it"
        End With
        .Refresh
    End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

axRibbon1.AllowToolTip = false;
EXRIBBONLib.Items var_Items = axRibbon1.Items;

```

```
var_Items.Add("Item",null,null).Tooltip = "this is a bit of text to be shown when  
cursor hovers it";  
var_Items.Add("Item",null,null).Tooltip = "this is a bit of text to be shown when  
cursor hovers it";  
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()  
{  
    COM com_Item,com_Items;  
    anytype var_Item,var_Items;  
    ;  
  
    super();  
  
    exribbon1.AllowToolTip(false);  
    var_Items = exribbon1.Items(); com_Items = var_Items;  
    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =  
var_Item;  
    com_Item.Tooltip("this is a bit of text to be shown when cursor hovers it");  
    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =  
var_Item;  
    com_Item.Tooltip("this is a bit of text to be shown when cursor hovers it");  
    exribbon1.Refresh();  
}
```

Delphi 8 (.NET only)

```
with AxRibbon1 do  
begin  
    AllowToolTip := False;  
    with Items do  
    begin  
        Add('Item',Nil,Nil).Tooltip := 'this is a bit of text to be shown when cursor hovers  
it';  
        Add('Item',Nil,Nil).Tooltip := 'this is a bit of text to be shown when cursor hovers
```

```
it';
    end;
    Refresh();
end
```

Delphi (standard)

```
with Ribbon1 do
begin
    AllowToolTip := False;
    with Items do
    begin
        Add('Item',Null,Null).Tooltip := 'this is a bit of text to be shown when cursor
hovers it';
        Add('Item',Null,Null).Tooltip := 'this is a bit of text to be shown when cursor
hovers it';
    end;
    Refresh();
end
```

VFP

```
with thisform.Ribbon1
.AllowToolTip = .F.
with .Items
    .Add("Item").Tooltip = "this is a bit of text to be shown when cursor hovers it"
    .Add("Item").Tooltip = "this is a bit of text to be shown when cursor hovers it"
endwith
.Refresh
endwith
```

dBASE Plus

```
local oRibbon,var_Item,var_Item1,var_Items

oRibbon = form.Activex1.nativeObject
oRibbon.AllowToolTip = false
var_Items = oRibbon.Items
```

```

// var_Items.Add("Item").Tooltip = "this is a bit of text to be shown when  

cursor hovers it"
var_Item = var_Items.Add("Item")
with (oRibbon)
    TemplateDef = [Dim var_Item]
    TemplateDef = var_Item
    Template = [var_Item.Tooltip = "this is a bit of text to be shown when cursor  

hovers it"]
endwith
// var_Items.Add("Item").Tooltip = "this is a bit of text to be shown when  

cursor hovers it"
var_Item1 = var_Items.Add("Item")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.Tooltip = "this is a bit of text to be shown when cursor  

hovers it"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
oRibbon.AllowToolTip = .f
var_Items = oRibbon.Items
' var_Items.Add("Item").Tooltip = "this is a bit of text to be shown when  

cursor hovers it"
var_Item = var_Items.Add("Item")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.Tooltip = \"this is a bit of text to be shown when

```

```
cursor hovers it\ ""
```

```
 ' var_Items.Add("Item").Tooltip = "this is a bit of text to be shown when  
cursor hovers it"
```

```
var_Item1 = var_Items.Add("Item")  
oRibbon.TemplateDef = "Dim var_Item1"  
oRibbon.TemplateDef = var_Item1  
oRibbon.Template = "var_Item1.Tooltip = \"this is a bit of text to be shown when  
cursor hovers it\ ""
```

```
oRibbon.Refresh()
```

Visual Objects

```
local var_Items as IItems
```

```
oDCOCX_Exontrol1:AllowToolTip := false
```

```
var_Items := oDCOCX_Exontrol1:Items
```

```
var_Items.Add("Item",nil,nil):Tooltip := "this is a bit of text to be shown when cursor  
hovers it"
```

```
var_Items.Add("Item",nil,nil):Tooltip := "this is a bit of text to be shown when cursor  
hovers it"
```

```
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Items
```

```
oRibbon = ole_1.Object
```

```
oRibbon.AllowToolTip = false
```

```
var_Items = oRibbon.Items
```

```
var_Items.Add("Item").Tooltip = "this is a bit of text to be shown when cursor  
hovers it"
```

```
var_Items.Add("Item").Tooltip = "this is a bit of text to be shown when cursor  
hovers it"
```

```
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
  Forward Send OnCreate
  Set ComAllowToolTip to False
  Variant voltems
  Get Comltems to voltems
  Handle holtems
  Get Create (RefClass(cComltems)) to holtems
  Set pvComObject of holtems to voltems
    Variant voltem
    Get ComAdd of holtems "Item" Nothing Nothing to voltem
    Handle holtem
    Get Create (RefClass(cComltem)) to holtem
    Set pvComObject of holtem to voltem
      Set ComTooltip of holtem to "this is a bit of text to be shown when cursor hovers it"
    Send Destroy to holtem
    Variant voltem1
    Get ComAdd of holtems "Item" Nothing Nothing to voltem1
    Handle holtem1
    Get Create (RefClass(cComltem)) to holtem1
    Set pvComObject of holtem1 to voltem1
      Set ComTooltip of holtem1 to "this is a bit of text to be shown when cursor hovers it"
    Send Destroy to holtem1
  Send Destroy to holtems
  Send ComRefresh
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
```

LOCAL oForm

LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

LOCAL oltems

LOCAL oRibbon

oForm := XbpDialog():new(AppDesktop())

oForm:drawingArea:clipChildren := .T.

oForm:create(„{100,100}, {640,480}„, .F.)

oForm:close := {|| PostAppEvent(xbeP_Quit)}

oRibbon := XbpActiveXControl():new(oForm:drawingArea)

oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/

oRibbon:create(„ {10,60},{610,370})

oRibbon:**AllowToolTip** := .F.

oltems := oRibbon:Items()

oltems:Add("Item"):Tooltip := "this is a bit of text to be shown when cursor
hovers it"

oltems:Add("Item"):Tooltip := "this is a bit of text to be shown when cursor
hovers it"

oRibbon:Refresh()

oForm:Show()

DO WHILE nEvent != xbeP_Quit

nEvent := AppEvent(@mp1, @mp2, @oXbp)

oXbp:handleEvent(nEvent, mp1, mp2)

ENDDO

RETURN

property Ribbon.Appearance as AppearanceEnum

Retrieves or sets the control's appearance.

Type	Description
AppearanceEnum	A AppearanceEnum expression that specifies the menu's frame appearance, or a color expression whose last 7 bits in the high significant byte of the value indicates the index of the skin in the Appearance collection, being displayed as control's borders. For instance, if the Appearance = 0x1000000, indicates that the first skin object in the Appearance collection defines the control's border. <i>The Client object in the skin, defines the client area of the control. 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</i>

By default, the Appearance property is ShadowBorder. The Appearance property specifies the menu's frame appearance. The [SelBackColor](#) property specifies the visual appearance of the item being selected / highlighted. The [BackColor](#) property specifies the control's background color. The [Background](#) property specifies the visual appearance for different parts of the control, including the radio-buttons, check-boxes or separator items. The [LocalAppearance](#) property specifies the visual appearance of the local popup. The [PopupAppearance](#) specifies a different visual appearance for the current submenu. When using EBN appearance, using the [PopupAppearance](#), [LocalAppearance](#) or Appearance, the distance between margins/borders and items client area is indicated by the client object of the skin/ebn object.

The following screen shot shows the control's frame as displayed by default:



The following screen shot shows the control's frame using a different EBN file:

☒ Expand

☐ Radio 1

☐ Radio 2

☐ Radio 3

☒ Expand

☐ Radio 1

☐ Radio 2

☒ Radio 3

method Ribbon.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 Ribbon1_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 Ribbon.BackColor as Color

Specifies the control's background color.

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

The BackColor property specifies the control's background color. Use the [PopupFlatBackColor](#) property to specify the background color of the left side of the control. The [ForeColor](#) property specifies the control's foreground color. The [SelBackColor](#) property specifies the visual appearance of the item being selected / highlighted. The [SelForeColor](#) property specifies the foreground color of the item being selected / highlighted. The [Background](#) property specifies the visual appearance for different parts of the control. The [Appearance](#) property specifies the menu's frame appearance. The [BackColor](#) property of the Item object specifies a different background color / visual appearance for the entire item.

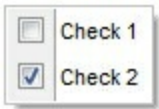
property Ribbon.Background(Part as BackgroundPartEnum)as Color

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

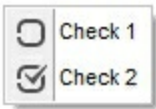
Type	Description
Part as BackgroundPartEnum	A BackgroundPartEnum expression that indicates the part to be changed
Color	A Color expression that indicates the background color for a specified part. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being 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.

Use the Background property to specify a different visual appearance for parts of the control, such as tooltip, check or radio buttons.

The following screen shot shows the check-boxes, as they are shown by default:



The following screen shot shows the check-boxes, as once a new visual appearance is applied:



How can I change the visual appearance of the check-boxes to be displayed in the ribbon control (ebn)?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .VisualAppearance
    .Add 1,"c:\exontrol\images\normal.ebn"
    .Add 2,"c:\exontrol\images\pushed.ebn"
  End With
  .Background(70) = &H1000000
  .Background(71) = &H2000000
```

```

With .Items
    With .Add("",2)
        .GroupPopup = 3 ' GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
    With .Items
        With .Add("Check 1")
            .Check = True
            .Checked = True
        End With
        .Add("Check 2").Check = True
    End With
End With
End With
End With
.Refresh
End With

```

VB6

```

With Ribbon1
    With .VisualAppearance
        .Add 1,"c:\exontrol\images\normal.ebn"
        .Add 2,"c:\exontrol\images\pushed.ebn"
    End With
    .Background(exCheckBoxState0) = &H1000000
    .Background(exCheckBoxState1) = &H2000000
    With .Items
        With .Add("",2)
            .GroupPopup = GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
        With .Items
            With .Add("Check 1")
                .Check = True
                .Checked = True
            End With
            .Add("Check 2").Check = True
        End With
    End With
End With

```

```
End With
.Refresh
End With
```

VB.NET

```
With Exribbon1
  With .VisualAppearance
    .Add(1,"c:\exontrol\images\normal.ebn")
    .Add(2,"c:\exontrol\images\pushed.ebn")
  End With

  .set_Background32(exontrol.EXRIBBONLib.BackgroundPartEnum.exCheckBoxState0,&H

  .set_Background32(exontrol.EXRIBBONLib.BackgroundPartEnum.exCheckBoxState1,&H

  With .Items
    With .Add("",2)
      .GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup
    With .Items
      With .Add("Check 1")
        .Check = True
        .Checked = True
      End With
      .Add("Check 2").Check = True
    End With
  End With
End With
.Refresh()
End With
```

VB.NET for /COM

```
With AxRibbon1
  With .VisualAppearance
```

```

.Add(1,"c:\exontrol\images\normal.ebn")
.Add(2,"c:\exontrol\images\pushed.ebn")
End With

.set_Background(EXRIBBONLib.BackgroundPartEnum.exCheckBoxState0,16777216)

.set_Background(EXRIBBONLib.BackgroundPartEnum.exCheckBoxState1,33554432)
With .Items
    With .Add("",2)
        .GroupPopup = EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or
EXRIBBONLib.GroupPopupEnum.exGroupPopup
        With .Items
            With .Add("Check 1")
                .Check = True
                .Checked = True
            End With
            .Add("Check 2").Check = True
        End With
    End With
End With
.Refresh()
End With

```

C++

```

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

#import <ExRibbon.dll>
using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IAppearancePtr var_Appearance = spRibbon1-
>GetVisualAppearance();

```



```

var_Appearance->Add(1,"c:\\exontrol\\images\\normal.ebn");
var_Appearance->Add(2,"c:\\exontrol\\images\\pushed.ebn");
spRibbon1->PutBackground(EXRIBBONLib::exCheckBoxState0,0x1000000);
spRibbon1->PutBackground(EXRIBBONLib::exCheckBoxState1,0x2000000);
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"",long(2),vtMissing);
var_Item-
>PutGroupPopup(EXRIBBONLib::GroupPopupEnum(EXRIBBONLib::exNoGroupPopupFrame | EXRIBBONLib::exGroupPopup));
EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();
EXRIBBONLib::IItemPtr var_Item1 = var_Items1->Add(L"Check
1",vtMissing,vtMissing);
var_Item1->PutCheck(VARIANT_TRUE);
var_Item1->PutChecked(VARIANT_TRUE);
var_Items1->Add(L"Check 2",vtMissing,vtMissing)-
>PutCheck(VARIANT_TRUE);
spRibbon1->Refresh();

```

C++ Builder

```

Exribbonlib_tlb::IAppearancePtr var_Appearance = Ribbon1->VisualAppearance;
var_Appearance->Add(1,TVariant("c:\\exontrol\\images\\normal.ebn"));
var_Appearance->Add(2,TVariant("c:\\exontrol\\images\\pushed.ebn"));
Ribbon1->Background[Exribbonlib_tlb::BackgroundPartEnum::exCheckBoxState0] =
0x1000000;
Ribbon1->Background[Exribbonlib_tlb::BackgroundPartEnum::exCheckBoxState1] =
0x2000000;
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
Exribbonlib_tlb::IItemPtr var_Item = var_Items->Add(L"",TVariant(2),TNoParam());
var_Item->GroupPopup =
Exribbonlib_tlb::GroupPopupEnum::exNoGroupPopupFrame |
Exribbonlib_tlb::GroupPopupEnum::exGroupPopup;
Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;
Exribbonlib_tlb::IItemPtr var_Item1 = var_Items1->Add(L"Check
1",TNoParam(),TNoParam());
var_Item1->Check = true;

```

```

        var_Item1->Checked = true;
        var_Items1->Add(L"Check 2",TNoParam(),TNoParam())->Check = true;
        Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Appearance var_Appearance = exribbon1.VisualAppearance;
    var_Appearance.Add(1,"c:\\exontrol\\images\\normal.ebn");
    var_Appearance.Add(2,"c:\\exontrol\\images\\pushed.ebn");
exribbon1.set_Background32(exontrol.EXRIBBONLib.BackgroundPartEnum.exCheckBox

exribbon1.set_Background32(exontrol.EXRIBBONLib.BackgroundPartEnum.exCheckBox

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
    exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
        var_Item.GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup;
        exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
            exontrol.EXRIBBONLib.Item var_Item1 = var_Items1.Add("Check 1",null,null);
                var_Item1.Check = true;
                var_Item1.Checked = true;
            var_Items1.Add("Check 2",null,null).Check = true;
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload= 'Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Appearance = Ribbon1.VisualAppearance;
        var_Appearance.Add(1,"c:\\exontrol\\images\\normal.ebn");

```

```

var Appearance.Add(2,"c:\exontrol\images\pushed.ebn");
Ribbon1.Background(70) = 16777216;
Ribbon1.Background(71) = 33554432;
var var_Items = Ribbon1.Items;
var var_Item = var_Items.Add("",2,null);
var_Item.GroupPopup = 3;
var var_Items1 = var_Item.Items;
var var_Item1 = var_Items1.Add("Check 1",null,null);
var_Item1.Check = true;
var_Item1.Checked = true;
var_Items1.Add("Check 2",null,null).Check = true;
Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
With Ribbon1
With VisualAppearance
.Add 1,"c:\exontrol\images\normal.ebn"
.Add 2,"c:\exontrol\images\pushed.ebn"
End With
.Background(70) = &H1000000
.Background(71) = &H2000000
With .Items
With .Add("",2)
.GroupPopup = 3 ' GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
With .Items

```

```

        With .Add("Check 1")
            .Check = True
            .Checked = True
        End With
        .Add("Check 2").Check = True
    End With
End With
End With
.Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Appearance var_Appearance = axRibbon1.VisualAppearance;
var_Appearance.Add(1,"c:\\exontrol\\images\\normal.ebn");
var_Appearance.Add(2,"c:\\exontrol\\images\\pushed.ebn");
axRibbon1.set_Background(EXRIBBONLib.BackgroundPartEnum.exCheckBoxState0,0x00000000);

axRibbon1.set_Background(EXRIBBONLib.BackgroundPartEnum.exCheckBoxState1,0x00000000);

EXRIBBONLib.Items var_Items = axRibbon1.Items;
EXRIBBONLib.Item var_Item = var_Items.Add("",2,null);
var_Item.GroupPopup =
EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
EXRIBBONLib.GroupPopupEnum.exGroupPopup;
EXRIBBONLib.Items var_Items1 = var_Item.Items;
EXRIBBONLib.Item var_Item1 = var_Items1.Add("Check 1",null,null);
var_Item1.Check = true;
var_Item1.Checked = true;
var_Items1.Add("Check 2",null,null).Check = true;
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Appearance,com_Item,com_Item1,com_Item2,com_Items,com_Items1;
    anytype var_Appearance,var_Item,var_Item1,var_Item2,var_Items,var_Items1;
    ;

    super();

    var_Appearance = exribbon1.VisualAppearance(); com_Appearance =
var_Appearance;
    com_Appearance.Add(1,"c:\\exontrol\\images\\normal.ebn");
    com_Appearance.Add(2,"c:\\exontrol\\images\\pushed.ebn");
    exribbon1.Background(70/*exCheckBoxState0*/,0x1000000);
    exribbon1.Background(71/*exCheckBoxState1*/,0x2000000);
    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("",COMVariant::createFromInt(2)); com_Item =
var_Item;
    com_Item.GroupPopup(3/*exNoGroupPopupFrame | exGroupPopup*/);
    var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
    var_Item1 = com_Items1.Add("Check 1"); com_Item1 = var_Item1;
    com_Item1.Check(true);
    com_Item1.Checked(true);
    var_Item2 = COM::createFromObject(com_Items1.Add("Check 2"));
com_Item2 = var_Item2;
    com_Item2.Check(true);
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
    with VisualAppearance do
    begin
        Add(1,'c:\\exontrol\\images\\normal.ebn');
        Add(2,'c:\\exontrol\\images\\pushed.ebn');
    end;
end;

```

```

set_Background(EXRIBBONLib.BackgroundPartEnum.exCheckBoxState0,$1000000);

set_Background(EXRIBBONLib.BackgroundPartEnum.exCheckBoxState1,$2000000);
  with Items do
  begin
    with Add('',TObject(2),Nil) do
    begin
      GroupPopup :=
Integer(EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib.GroupPopupEnum.exGroupPopup);
      with Items do
      begin
        with Add('Check 1',Nil,Nil) do
        begin
          Check := True;
          Checked := True;
        end;
        Add('Check 2',Nil,Nil).Check := True;
      end;
    end;
  end;
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  with VisualAppearance do
  begin
    Add(1,'c:\exontrol\images\normal.ebn');
    Add(2,'c:\exontrol\images\pushed.ebn');
  end;
  Background[EXRIBBONLib_TLB.exCheckBoxState0] := $1000000;
  Background[EXRIBBONLib_TLB.exCheckBoxState1] := $2000000;
  with Items do

```

```

begin
  with Add('',OleVariant(2),Null) do
    begin
      GroupPopup := Integer(EXRIBBONLib_TLB.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib_TLB.exGroupPopup);
      with Items do
        begin
          with Add('Check 1',Null,Null) do
            begin
              Check := True;
              Checked := True;
            end;
            Add('Check 2',Null,Null).Check := True;
          end;
        end;
      end;
    end;
  Refresh();
end

```

VFP

```

with thisform.Ribbon1
  with .VisualAppearance
    .Add(1,"c:\exontrol\images\normal.ebn")
    .Add(2,"c:\exontrol\images\pushed.ebn")
  endwith
  .Object.Background(70) = 0x1000000
  .Object.Background(71) = 0x2000000
  with .Items
    with .Add("",2)
      .GroupPopup = 3 && GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
      with .Items
        with .Add("Check 1")
          .Check = .T.
          .Checked = .T.
        endwith
      endwith
    endwith
  endwith

```

```

        .Add("Check 2").Check = .T.
    endwhile
endwith
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Appearance,var_Item,var_Item1,var_Item2,var_Items,var_Items1

oRibbon = form.ActiveX1.nativeObject
var_Appearance = oRibbon.VisualAppearance
    var_Appearance.Add(1,"c:\exontrol\images\normal.ebn")
    var_Appearance.Add(2,"c:\exontrol\images\pushed.ebn")
oRibbon.Template = [Background(70) = 0x1000000] // oRibbon.Background(70) = 0x1000000
oRibbon.Template = [Background(71) = 0x2000000] // oRibbon.Background(71) = 0x2000000
var_Items = oRibbon.Items
    var_Item = var_Items.Add("",2)
    var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
    var_Items1 = var_Item.Items
        var_Item1 = var_Items1.Add("Check 1")
        var_Item1.Check = true
        var_Item1.Checked = true
        // var_Items1.Add("Check 2").Check = true
    var_Item2 = var_Items1.Add("Check 2")
    with (oRibbon)
        TemplateDef = [Dim var_Item2]
        TemplateDef = var_Item2
        Template = [var_Item2.Check = true]
    endwhile
oRibbon.Refresh()

```

XBasic (Alpha Five)


```

Dim oRibbon as P
Dim var_Appearance as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Item2 as P
Dim var_Items as P
Dim var_Items1 as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Appearance = oRibbon.VisualAppearance
    var_Appearance.Add(1,"c:\exontrol\images\normal.ebn")
    var_Appearance.Add(2,"c:\exontrol\images\pushed.ebn")
oRibbon.Template = "Background(70) = 16777216" ' oRibbon.Background(70) = 16777216
oRibbon.Template = "Background(71) = 33554432" ' oRibbon.Background(71) = 33554432
var_Items = oRibbon.Items
    var_Item = var_Items.Add("",2)
    var_Item.GroupPopup = 3 ' exNoGroupPopupFrame + exGroupPopup
    var_Items1 = var_Item.Items
        var_Item1 = var_Items1.Add("Check 1")
        var_Item1.Check = .t.
        var_Item1.Checked = .t.
        ' var_Items1.Add("Check 2").Check = .t.
    var_Item2 = var_Items1.Add("Check 2")
    oRibbon.TemplateDef = "Dim var_Item2"
    oRibbon.TemplateDef = var_Item2
    oRibbon.Template = "var_Item2.Check = True"

oRibbon.Refresh()

```

Visual Objects

```

local var_Appearance as IAppearance
local var_Item,var_Item1 as IItem
local var_Items,var_Items1 as IItems

```

```

var_Appearance := oDCOCX_Exontrol1:VisualAppearance
  var_Appearance:Add(1,"c:\exontrol\images\normal.ebn")
  var_Appearance:Add(2,"c:\exontrol\images\pushed.ebn")
oDCOCX_Exontrol1:[Background,exCheckBoxState0] := 0x1000000
oDCOCX_Exontrol1:[Background,exCheckBoxState1] := 0x2000000
var_Items := oDCOCX_Exontrol1:Items
  var_Item := var_Items:Add("",2,nil)
  var_Item:GroupPopup := exNoGroupPopupFrame | exGroupPopup
  var_Items1 := var_Item:Items
    var_Item1 := var_Items1:Add("Check 1",nil,nil)
    var_Item1:Check := true
    var_Item1:Checked := true
    var_Items1:Add("Check 2",nil,nil):Check := true
oDCOCX_Exontrol1:Refresh()

```

PowerBuilder

```

OleObject oRibbon,var_Appearance,var_Item,var_Item1,var_Items,var_Items1

oRibbon = ole_1.Object
var_Appearance = oRibbon.VisualAppearance
  var_Appearance.Add(1,"c:\exontrol\images\normal.ebn")
  var_Appearance.Add(2,"c:\exontrol\images\pushed.ebn")
oRibbon.Background(70,16777216 /*0x1000000*/)
oRibbon.Background(71,33554432 /*0x2000000*/)
var_Items = oRibbon.Items
  var_Item = var_Items.Add("",2)
  var_Item.GroupPopup = 3 /*exNoGroupPopupFrame | exGroupPopup*/
  var_Items1 = var_Item.Items
    var_Item1 = var_Items1.Add("Check 1")
    var_Item1.Check = true
    var_Item1.Checked = true
    var_Items1.Add("Check 2").Check = true
oRibbon.Refresh()

```

Procedure OnCreate

Forward Send OnCreate

Variant voAppearance

Get **ComVisualAppearance** to voAppearance

Handle hoAppearance

Get Create (RefClass(cComAppearance)) to hoAppearance

Set pvComObject of hoAppearance to voAppearance

Get ComAdd of hoAppearance 1 "c:\exontrol\images\normal.ebn" to Nothing

Get ComAdd of hoAppearance 2 "c:\exontrol\images\pushed.ebn" to Nothing

Send Destroy to hoAppearance

Set **ComBackground** OLEexCheckBoxState0 to |CI\$1000000

Set **ComBackground** OLEexCheckBoxState1 to |CI\$2000000

Variant voltems

Get ComItems to voltems

Handle holtems

Get Create (RefClass(cComItems)) to holtems

Set pvComObject of holtems to voltems

Variant voltem

Get ComAdd of holtems "" 2 Nothing to voltem

Handle holtem

Get Create (RefClass(cComItem)) to holtem

Set pvComObject of holtem to voltem

Set ComGroupPopup of holtem to (OLEexNoGroupPopupFrame +
OLEexGroupPopup)

Variant voltems1

Get ComItems of holtem to voltems1

Handle holtems1

Get Create (RefClass(cComItems)) to holtems1

Set pvComObject of holtems1 to voltems1

Variant voltem1

Get ComAdd of holtems1 "Check 1" Nothing Nothing to voltem1

Handle holtem1

Get Create (RefClass(cComItem)) to holtem1

Set pvComObject of holtem1 to voltem1

Set ComCheck of holtem1 to True

```

        Set ComChecked of holtem1 to True
    Send Destroy to holtem1
    Variant voltem2
    Get ComAdd of holtems1 "Check 2" Nothing Nothing to voltem2
    Handle holtem2
    Get Create (RefClass(cComItem)) to holtem2
    Set pvComObject of holtem2 to voltem2
        Set ComCheck of holtem2 to True
    Send Destroy to holtem2
    Send Destroy to holtems1
    Send Destroy to holtem
    Send Destroy to holtems
    Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oAppearance
    LOCAL oltem,oltem1
    LOCAL oltems,oltems1
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

```

```
oAppearance := oRibbon:VisualAppearance()
  oAppearance:Add(1,"c:\exontrol\images\normal.ebn")
  oAppearance:Add(2,"c:\exontrol\images\pushed.ebn")
oRibbon:SetProperty("Background",70/*exCheckBoxState0*/,0x1000000)
oRibbon:SetProperty("Background",71/*exCheckBoxState1*/,0x2000000)
oltems := oRibbon:Items()
  oltem := oltems:Add("",2)
    oltem:GroupPopup := 3/*exNoGroupPopupFrame+exGroupPopup*/
    oltems1 := oltem:Items()
      oltem1 := oltems1:Add("Check 1")
        oltem1:Check := .T.
        oltem1:Checked := .T.
      oltems1:Add("Check 2"):Check := .T.
oRibbon:Refresh()
```

```
oForm:Show()
DO WHILE nEvent != xbeP_Quit
  nEvent := AppEvent( @mp1, @mp2, @oXbp )
  oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN
```

property Ribbon.Cursor as Variant

Gets or sets the cursor that is displayed when the mouse pointer hovers the control.

Type	Description
Variant	A String expression that defines the cursor to be shown when the cursor hovers the menu. The Valid values are listed bellow. Also the Cursor property could point to a cursor file to be loaded and shown while the cursor hovers the ribbon.

By default, the Cursor property is "exDefault". Use the Cursor property to specify a different cursor when it hovers the menu control. Use the [Cursor](#) property of the Item object to specify a different cursor when it hovers the item only.

The supported values are:

- "exDefault", Standard arrow
- "exArrow", Standard arrow
- "exCross", Crosshair
- "exIBeam", I-beam
- "exIcon", Reserved
- "exSize", Reserved, use the "exSizeAll"
- "exSizeNESW", Double-pointed arrow pointing northeast and southwest
- "exSizeNS", Double-pointed arrow pointing north and south
- "exSizeNWSE", Double-pointed arrow pointing northwest and southeast
- "exSizeWE", Double-pointed arrow pointing west and east
- "exUpArrow", Vertical arrow
- "exHourglass", Hourglass
- "exNoDrop", Slashed circle
- "exArrowHourglass"
- "exHelp", Arrow and question mark
- "exSizeAll", Four-pointed arrow pointing north, south, east, and west
- "exHand", Hand

Any other value indicates the path to a cursor file to be displayed when the pointer hovers the menu control.

Can I change the cursor where it hovers the item?

VBA (MS Access, Excell...)

```
With Ribbon1
    .Cursor = "exCross"
```

```
With .Items
    .Add "Item 1"
    .Add("Item 2").Cursor = "exNoDrop"
End With
.Refresh
End With
```

VB6

```
With Ribbon1
    .Cursor = "exCross"
    With .Items
        .Add "Item 1"
        .Add("Item 2").Cursor = "exNoDrop"
    End With
    .Refresh
End With
```

VB.NET

```
With Exribbon1
    .Cursor = "exCross"
    With .Items
        .Add("Item 1")
        .Add("Item 2").Cursor = "exNoDrop"
    End With
    .Refresh()
End With
```

VB.NET for /COM

```
With AxRibbon1
    .Cursor = "exCross"
    With .Items
        .Add("Item 1")
        .Add("Item 2").Cursor = "exNoDrop"
    End With
    .Refresh()
```

C++

```

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

    #import <ExRibbon.dll>
    using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
spRibbon1->PutCursor("exCross");
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    var_Items->Add(L"Item 1",vtMissing,vtMissing);
    var_Items->Add(L"Item 2",vtMissing,vtMissing)->PutCursor("exNoDrop");
spRibbon1->Refresh();

```

C++ Builder

```

Ribbon1->set_Cursor(TVariant("exCross"));
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    var_Items->Add(L"Item 1",TNoParam(),TNoParam());
    var_Items->Add(L"Item 2",TNoParam(),TNoParam())-
>set_Cursor(TVariant("exNoDrop"));
Ribbon1->Refresh();

```

C#

```

exribbon1.Cursor = "exCross";
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
    var_Items.Add("Item 1",null,null);
    var_Items.Add("Item 2",null,null).Cursor = "exNoDrop";
exribbon1.Refresh();

```


JScript/JavaScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Ribbon1.Cursor = "exCross";
    var var_Items = Ribbon1.Items;
    var_Items.Add("Item 1",null,null);
    var_Items.Add("Item 2",null,null).Cursor = "exNoDrop";
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        .Cursor = "exCross"
        With .Items
            .Add "Item 1"
            .Add("Item 2").Cursor = "exNoDrop"
        End With
        .Refresh
    End With
End Function
```

```
</SCRIPT>  
</BODY>
```

C# for /COM

```
axRibbon1.Cursor = "exCross";  
EXRIBBONLib.Items var_Items = axRibbon1.Items;  
    var_Items.Add("Item 1",null,null);  
    var_Items.Add("Item 2",null,null).Cursor = "exNoDrop";  
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()  
{  
    COM com_Item,com_Items;  
    anytype var_Item,var_Items;  
    ;  
  
    super();  
  
    exribbon1.Cursor("exCross");  
    var_Items = exribbon1.Items(); com_Items = var_Items;  
        com_Items.Add("Item 1");  
        var_Item = COM::createFromObject(com_Items.Add("Item 2")); com_Item =  
var_Item;  
        com_Item.Cursor("exNoDrop");  
    exribbon1.Refresh();  
}
```

Delphi 8 (.NET only)

```
with AxRibbon1 do  
begin  
    Cursor := 'exCross';  
    with Items do
```

```

begin
  Add('Item 1',Nil,Nil);
  Add('Item 2',Nil,Nil).Cursor := 'exNoDrop';
end;
Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  Cursor := 'exCross';
  with Items do
  begin
    Add('Item 1',Null,Null);
    Add('Item 2',Null,Null).Cursor := 'exNoDrop';
  end;
  Refresh();
end

```

VFP

```

with thisform.Ribbon1
.Cursor = "exCross"
with .Items
.Add("Item 1")
.Add("Item 2").Cursor = "exNoDrop"
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Items

oRibbon = form.ActiveX1.nativeObject
oRibbon.Cursor = "exCross"
var_Items = oRibbon.Items

```

```

var_Items.Add("Item 1")
// var_Items.Add("Item 2").Cursor = "exNoDrop"
var_Item = var_Items.Add("Item 2")
with (oRibbon)
    TemplateDef = [Dim var_Item]
    TemplateDef = var_Item
    Template = [var_Item.Cursor = "exNoDrop"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
oRibbon.Cursor = "exCross"
var_Items = oRibbon.Items
var_Items.Add("Item 1")
' var_Items.Add("Item 2").Cursor = "exNoDrop"
var_Item = var_Items.Add("Item 2")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.Cursor = \"exNoDrop\""

oRibbon.Refresh()

```

Visual Objects

```

local var_Items as IItems

oDCOCX_Exontrol1:Cursor := "exCross"
var_Items := oDCOCX_Exontrol1.Items
var_Items.Add("Item 1",nil,nil)
var_Items.Add("Item 2",nil,nil):Cursor := "exNoDrop"

```

```
oDCOCX_Exontrol1:Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Items  
  
oRibbon = ole_1.Object  
oRibbon.Cursor = "exCross"  
var_Items = oRibbon.Items  
    var_Items.Add("Item 1")  
    var_Items.Add("Item 2").Cursor = "exNoDrop"  
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate  
    Forward Send OnCreate  
    Set ComCursor to "exCross"  
    Variant voltems  
    Get ComItems to voltems  
    Handle holtems  
    Get Create (RefClass(cComItems)) to holtems  
    Set pvComObject of holtems to voltems  
        Get ComAdd of holtems "Item 1" Nothing Nothing to Nothing  
        Variant voltem  
        Get ComAdd of holtems "Item 2" Nothing Nothing to voltem  
        Handle holtem  
        Get Create (RefClass(cComItem)) to holtem  
        Set pvComObject of holtem to voltem  
            Set ComCursor of holtem to "exNoDrop"  
        Send Destroy to holtem  
    Send Destroy to holtems  
    Send ComRefresh  
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
    LOCAL oForm
```

```
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
    LOCAL oltems
```

```
    LOCAL oRibbon
```

```
    oForm := XbpDialog():new( AppDesktop() )
```

```
    oForm:drawingArea:clipChildren := .T.
```

```
    oForm:create( ,, {100,100}, {640,480},,, .F. )
```

```
    oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
```

```
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-  
CFBE431702E2}*/
```

```
    oRibbon:create(,, {10,60},{610,370} )
```

```
    oRibbon:Cursor := "exCross"
```

```
    oltems := oRibbon:Items()
```

```
        oltems:Add("Item 1")
```

```
        oltems:Add("Item 2"):Cursor := "exNoDrop"
```

```
    oRibbon:Refresh()
```

```
    oForm:Show()
```

```
    DO WHILE nEvent != xbeP_Quit
```

```
        nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
        oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
    ENDDO
```

```
RETURN
```

property Ribbon.Debug as Boolean

Retrieves or sets a value that indicating whether the item's identifier is visible.

Type	Description
Boolean	A Boolean expression that specifies whether the identifiers of the items

By default, the Debug property is False. Use the Debug property to display the identifiers for all visible items, for debugging purposes. The First number in the [] parenthesis indicates the item's [ID](#) property.

property Ribbon.Enabled as Boolean

Enables or disables the control.

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

By default, the Enabled property is True. Use the Enabled property to disable the control. When the control is disabled the inside elements look as grayed. Use the [Enabled](#) property of the Item object to disable a specific item. Use the [Locked](#) property to disabled the control without changing the visual appearance of the inside elements.

How can I enable or disable an item?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    .Add("Item").Enabled = False
    .Add("").ToString = "Item[dis]"
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items
    .Add("Item").Enabled = False
    .Add("").ToString = "Item[dis]"
  End With
  .Refresh
End With
```

VB.NET

```
With Exribbon1
  With .Items
    .Add("Item").Enabled = False
    .Add("").ToString = "Item[dis]"
  End With
End With
```



```
End With  
.Refresh()  
End With
```

VB.NET for /COM

```
With AxRibbon1  
    With .Items  
        .Add("Item").Enabled = False  
        .Add("").ToString = "Item[dis]"  
    End With  
    .Refresh()  
End With
```

C++

```
/*  
    Copy and paste the following directives to your header file as  
    it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control  
    Library'  
  
    #import <ExRibbon.dll>  
    using namespace EXRIBBONLib;  
*/  
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-  
>GetControlUnknown();  
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();  
var_Items->Add(L"Item",vtMissing,vtMissing)->PutEnabled(VARIANT_FALSE);  
var_Items->Add(L"",vtMissing,vtMissing)->PutToString(L"Item[dis]");  
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;  
var_Items->Add(L"Item",TNoParam(),TNoParam())->Enabled = false;  
var_Items->Add(L"",TNoParam(),TNoParam())->ToString = L"Item[dis]";  
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
var_Items.Add("Item",null,null).Enabled = false;
var_Items.Add("",null,null).ToString = "Item[dis]";
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var_Items.Add("Item",null,null).Enabled = false;
    var_Items.Add("",null,null).ToString = "Item[dis]";
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
```

```

        .Add("Item").Enabled = False
        .Add("").ToString = "Item[dis]"
    End With
    .Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.Add("Item",null,null).Enabled = false;
var_Items.Add("",null,null).ToString = "Item[dis]";
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
    com_Item.Enabled(false);
    var_Item = COM::createFromObject(com_Items.Add("")); com_Item = var_Item;
    com_Item.ToString("Item[dis]");
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
  with Items do
  begin
    Add('Item',Nil,Nil).Enabled := False;
    Add('',Nil,Nil).ToString := 'Item[dis]';
  end;
  Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  with Items do
  begin
    Add('Item',Null,Null).Enabled := False;
    Add('',Null,Null).ToString := 'Item[dis]';
  end;
  Refresh();
end

```

VFP

```

with thisform.Ribbon1
  with .Items
    .Add("Item").Enabled = .F.
    .Add("").ToString = "Item[dis]"
  endwith
  .Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Items

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items

```

```

// var_Items.Add("Item").Enabled = false
var_Item = var_Items.Add("Item")
with (oRibbon)
    TemplateDef = [Dim var_Item]
    TemplateDef = var_Item
    Template = [var_Item.Enabled = false]
endwith
// var_Items.Add("").ToString = "Item[dis]"
var_Item1 = var_Items.Add("")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.ToString = "Item[dis]"]
endwith
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Items as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
' var_Items.Add("Item").Enabled = .f.
var_Item = var_Items.Add("Item")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.Enabled = False"

' var_Items.Add("").ToString = "Item[dis]"
var_Item1 = var_Items.Add("")
oRibbon.TemplateDef = "Dim var_Item1"
oRibbon.TemplateDef = var_Item1
oRibbon.Template = "var_Item1.ToString = \"Item[dis]\""

```

```
oRibbon.Refresh()
```

Visual Objects

```
local var_Items as IItems  
  
var_Items := oDCOCX_Exontrol1:Items  
var_Items:Add("Item",nil,nil):Enabled := false  
var_Items:Add("",nil,nil):ToString := "Item[dis]"  
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Items  
  
oRibbon = ole_1.Object  
var_Items = oRibbon.Items  
var_Items.Add("Item").Enabled = false  
var_Items.Add("").ToString = "Item[dis]"  
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate  
Forward Send OnCreate  
Variant voltems  
Get ComItems to voltems  
Handle holtems  
Get Create (RefClass(cComItems)) to holtems  
Set pvComObject of holtems to voltems  
Variant voltem  
Get ComAdd of holtems "Item" Nothing Nothing to voltem  
Handle holtem  
Get Create (RefClass(cComItem)) to holtem
```

```

Set pvComObject of holtem to voltem
  Set ComEnabled of holtem to False
Send Destroy to holtem
Variant voltem1
Get ComAdd of holtems "" Nothing Nothing to voltem1
Handle holtem1
Get Create (RefClass(cComItem)) to holtem1
Set pvComObject of holtem1 to voltem1
  Set ComToString of holtem1 to "Item[dis]"
Send Destroy to holtem1
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oltems
  LOCAL oRibbon

  oForm := XbpDialog():new( AppDesktop() )
  oForm:drawingArea:clipChildren := .T.
  oForm:create( ,, {100,100}, {640,480},,, .F. )
  oForm:close := {|| PostAppEvent( xbeP_Quit )}

  oRibbon := XbpActiveXControl():new( oForm:drawingArea )
  oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
  oRibbon:create(,, {10,60},{610,370} )

  oltems := oRibbon:Items()
  oltems:Add("Item"):Enabled := .F.

```

```
    oItems:Add("").ToString := "Item[dis]"
oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN
```


property Ribbon.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 Ribbon.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 String 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).

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 Ribbon.Font as IFontDisp

Retrieves or sets the control's font.

Type	Description
IFontDisp	A Font object to be used to shown the control items.

Use the Font property to specify a different font to show the items in the ribbon. The Font's height controls the height of the items in the control. You can use the HTML tag to specify a different font for a specified item in the [Caption](#) property. The [ForeColor](#) property of the control specifies the foreground color of items in the control.

property Ribbon.ForeColor as Color

Specifies the control's foreground color.

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

The ForeColor property specifies the control's foreground color. The [BackColor](#) property specifies the control's background color. The [SelBackColor](#) property specifies the visual appearance of the item being selected / highlighted. The [SelForeColor](#) property specifies the foreground color of the item being selected / highlighted. The [Background](#) property specifies the visual appearance for different parts of the control. The [Appearance](#) property specifies the menu's frame appearance. You can use the <fgcolor> HTML tag to specify a different foreground colot for a specified item in the [Caption](#) property. The [ForeColor](#) property of the Item object specifies a different foreground color for the entire item.

method **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 Expression.

The FormatABC method formats the A,B,C values based on the giving expression and returns the result. 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.

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 Expression of 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

The constants are (DPI-Aware components):

- **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 y-scale. For instance, if current DPI setting is 100%, the dpiy constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpiy returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%

The supported binary arithmetic operators are:

- * (multiplicity operator), priority 5
- / (divide operator), priority 5
- **mod** (remainder operator), priority 5
- + (addition operator), priority 4 (concatenates two strings, if one of the operands is of string type)
- - (subtraction operator), priority 4

The supported unary boolean operators are:

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

The supported binary boolean operators are:

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

The supported binary boolean operators, all these with the same priority 0, are :

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

The supported binary range operators, all these with the same priority 5, are :

- **MIN** (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.

- **MAX** (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 than *if* (immediate if operator) alternative.

- **case()** (case operator) returns and executes one of n expressions, depending on the evaluation of the expression (IIF - immediate IF operator is a binary case() operator). The syntax for case() operator is:

expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)

If the default part is missing, the case() operator returns the value of the expression if it is not found in the collection of cases (c1, c2, ...). For instance, if the value of expression is not any of c1, c2, the default_expression is executed and returned. If the value of the expression is c1, then the case() operator executes and returns the expression1. The default, c1, c2, c3, ... must be constant elements as numbers, dates or strings. For instance, the *date(shortdate(value)) case (default:0 ; #1/1/2002#:1 ; #2/1/2002#:1; #4/1/2002#:1; #5/1/2002#:1)* indicates that only #1/1/2002#, #2/1/2002#, #4/1/2002# and #5/1/2002# dates returns 1, since the others returns 0. For instance the following sample specifies the hour being non-working for specified dates: *date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)* statement indicates the working hours for dates as follows:

- #4/1/2009#, from hours 06:00 AM to 12:00 PM
- #4/5/2009#, from hours 07:00 AM to 10:00 AM and hours 03:00PM, 04:00PM, 06:00PM and 10:00PM
- #5/1/2009#, from hours 12:00 AM to 08:00 AM

The *in*, *switch* and *case()* use binary search to look for elements so they are faster 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. For instance *type(%1) = 8* specifies the cells (on the column 1) that contains string values.

Here's few predefined types:

- 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
- **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#

Other known 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

Other known 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"
- **startswith** (binary operator) specifies whether a string starts with specified string (0 if not found, -1 if found). For instance *"Mihai" startwith "Mi"* returns -1
- **endwith** (binary operator) specifies whether a string ends with specified string (0 if not found, -1 if found). For instance *"Mihai" endwith "ai"* returns -1
- **contains** (binary operator) specifies whether a string contains another specified string (0 if not found, -1 if found). For instance *"Mihai" contains "ha"* returns -1
- **left** (binary operator) retrieves the left part of the string. For instance *"Mihai" left 2* returns "Mi".
- **right** (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, 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.

Other known 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

property Ribbon.Get (Criteria as MenuItemTypeEnum) as Variant

Retrieves an array of Item objects that meet the criteria.

Type	Description
Criteria as MenuItemTypeEnum	A MenuItemTypeEnum expression that type of items to be retrieved.
Variant	A Safe-Array of Item objects being returned.

The Get method can be used to get a collection / safe array of Item objects with a specified characteristics. For instance, you can collect the items of Edit type, or items that displays an icon using the Image property. The [GetChecked](#) property gets a collection of checked items. The [GetUnchecked](#) property gets a collection of checked items. The [GetRadio](#) method gets a safe array with the radio-items being checked within a radio group. For instance, the [GetChecked](#) property is equivalent with the Get(exCheckBoxMenuItem + exCheckedMenuItem), or in other words all items with the [Check](#) and [Checked](#) properties on True. The result of the Get method indicates a Safe-Array of [Item](#) objects, which means that you can use the **for each** statement to enumerate the elements in the collection. The [ItemType](#) property is a read-only property that gets the type of the item.

property Ribbon.GetChecked as Variant

Retrieves an array of Item objects, that displays a check box which is checked.

Type	Description
Variant	A Safe-Array of Item objects that indicates the checked items in the control. The collection does include only items with the Check property set on True.

The GetChecked property gets a collection of checked items. The [GetUnchecked](#) property gets a collection of unchecked items. The [GetRadio](#) method gets a safe array with the radio-items being checked within a radio group. The [Check](#) property indicates whether the current item displays a check box. The [Checked](#) property specifies whether the item is checked or un-checked. The [Radio](#) property specifies whether the item displays a radio-button. The [RadioGroup](#) property specifies a group of radio-buttons. A radio group allows a single radio-item to be checked inside.

The following VB sample displays the caption of the items being checked in the control:

```
Dim c As Variant
For Each c In ribbon.GetChecked
    Debug.Print vbTab & c.Caption
Next
```

The following VB/NET sample displays the caption of the items being checked in the control:

```
Dim c As Object
For Each c In Exribbon1.GetChecked
    Debug.Print(vbTab & c.Caption)
Next
```

The following C# sample displays the caption of the items being checked in the control:

```
foreach (exontrol.EXRIBBONLib.Item i in exribbon1.GetChecked)
    System.Diagnostics.Debug.Print("\t" + i.Caption);
```

property Ribbon.GetRadio ([RadioGroup as Variant]) as Variant

Retrieves an array of Item objects of radio type in the same group, that are checked.

Type	Description
RadioGroup as Variant	A Long expression that specifies the radio-group being queried, or zero if you were not used any RadioGroup call.
Variant	A Safe-Array of Item objects that indicates the radio-checked items in the control. The collection does include only items with the Radio property set on True. The collection may contains zero or one element indicating the radio-item being checked in the specified radio group.

The GetRadio method gets a safe array with the radio-items being checked within a radio group. The [GetChecked](#) property gets a collection of checked items. The [GetUnchecked](#) property gets a collection of unchecked items. The [Check](#) property indicates whether the current item displays a check box. The [Checked](#) property specifies whether the item is checked or un-checked. The [Radio](#) property specifies whether the item displays a radio-button. The [RadioGroup](#) property specifies a group of radio-buttons. A radio group allows a single radio-item to be checked inside.

The following VB sample displays the caption of radio-item being checked (single radio-group, or the [RadioGroup](#) property has not been used to create ore groups) :

```
Dim c As Variant
For Each c In ribbon.GetRadio
    Debug.Print vbTab & c.Caption
Next
```

The following VB sample displays the caption of radio-item being checked in the radio-group with the identifier 100:

```
Dim c As Variant
For Each c In ribbon.GetRadio(100)
    Debug.Print vbTab & c.Caption
Next
```

The following VB/NET sample displays the caption of radio-item being checked (single radio-group, or the [RadioGroup](#) property has not been used to create ore groups) :

```
Dim c As Variant
```

```
For Each c In ribbon.GetRadio
    Debug.Print vbTab & c.Caption
Next
```

The following VB/NET sample displays the caption of radio-item being checked in the radio-group with the identifier 100:

```
Dim c As Variant
For Each c In ribbon.get_GetRadio(100)
    Debug.Print vbTab & c.Caption
Next
```

The following C# sample displays the caption of radio-item being checked (single radio-group, or the [RadioGroup](#) property has not been used to create ore groups) :

```
foreach (exontrol.EXRIBBONLib.Item i in exribbon1.GetRadio)
    System.Diagnostics.Debug.Print("\t" + i.Caption);
```

The following C# sample displays the caption of radio-item being checked in the radio-group with the identifier 100:

```
foreach (exontrol.EXRIBBONLib.Item i in exribbon1.get_GetRadio(100))
    System.Diagnostics.Debug.Print("\t" + i.Caption);
```

property Ribbon.GetUnchecked as Variant

Retrieves an array of Item objects, that displays a check box which is unchecked.

Type	Description
Variant	A Safe-Array of Item objects that indicates the checked items in the control. The collection does include only items with the Check property set on True.

The GetUnchecked property gets a collection of checked items. The [GetChecked](#) property gets a collection of checked items. The [GetRadio](#) method gets a safe array with the radio-items being checked within a radio group. The [Check](#) property indicates whether the current item displays a check box. The [Checked](#) property specifies whether the item is checked or un-checked. The [Radio](#) property specifies whether the item displays a radio-button. The [RadioGroup](#) property specifies a group of radio-buttons. A radio group allows a single radio-item to be checked inside.

The following VB sample displays the caption of the items being un-checked in the control:

```
Dim c As Variant
For Each c In ribbon.GetUnchecked
    Debug.Print vbTab & c.Caption
Next
```

The following VB/NET sample displays the caption of the items being un-checked in the control:

```
Dim c As Object
For Each c In Exribbon1.GetUnchecked
    Debug.Print(vbTab & c.Caption)
Next
```

The following C# sample displays the caption of the items being un-checked in the control:

```
foreach (exontrol.EXRIBBONLib.Item i in exribbon1.GetUnchecked)
    System.Diagnostics.Debug.Print("\t" + i.Caption);
```

property Ribbon.HTMLPicture(Key as String) as Variant

Adds or replaces a picture in HTML captions.

Type	Description
Key as String	A String expression that indicates the key of the picture being added or replaced. If the Key property is Empty string, the entire collection of pictures is cleared.
Variant	<p>The HTMLPicture specifies the picture being associated to a key. It can be one of the followings:</p> <ul style="list-style-type: none">• a string expression that indicates the path to the picture file, being loaded.• a string expression that indicates the base64 encoded string that holds a picture object, Use the eximages tool to save your picture as base64 encoded format.• A Picture object that indicates the picture being added or replaced. (A Picture object implements IPicture interface), <p>If empty, the picture being associated to a key is removed. If the key already exists the new picture is replaced. If the key is not empty, and it doesn't not exist a new picture is added</p>

The HTMLPicture property handles a collection of custom size picture being displayed in the HTML captions, using the tags. By default, the HTMLPicture collection is empty. Use the HTMLPicture property to add new pictures to be used in HTML captions. For instance, the HTMLPicture("pic1") = "c:\winnt\zapotec.bmp", loads the zapotec picture and associates the pic1 key to it. Any "pic1" sequence in HTML captions, displays the pic1 picture. On return, the HTMLPicture property retrieves a Picture object (this implements the IPictureDisp interface). The tag can be used in the [Caption](#) property of the [Item](#) object. Use the [HTMLImage](#) property to assign a BMP, JPG, GIF or PNG file to left side of the caption, the same way as you will do with the [Image](#) property. Use the [PopupFlatImageWidth](#) property to specify the width of the column that displays icons/images/check or radio buttons.

property Ribbon.hWnd as Long

Retrieves the control's window handle.

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

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

method Ribbon.Images (Handle as Variant)

Sets the control's image list at runtime.

Type	Description
Handle as Variant	<p>The Handle parameter can be:</p> <ul style="list-style-type: none">• 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 (<i>string, loads the icon using its path</i>)• 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..." (<i>string, loads icons using base64 encoded string</i>)• A reference to a Microsoft ImageList control (mscomctl.ocx, MSComctlLib.ImageList type) that holds the icons to add (<i>object, loads icons from a Microsoft ImageList control</i>)• 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 (<i>object, loads icon from a Picture object</i>)• A long expression that identifies a handle to an Image List Control (the Handle should be of HIMAGELIST type). On 64-bit platforms, the Handle parameter must be a Variant of LongLong / LONG_PTR data type (signed 64-bit (8-byte) integers), saved under lVal field, as VT_I8 type. The LONGLONG / LONG_PTR is __int64, a 64-bit integer. For instance, in C++ you can use as Images(COleVariant((LONG_PTR)hImageList)) or Images(COleVariant((LONGLONG)hImageList)), where hImageList is of

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

The user can add images at design time, by drag and drop files to combo's image holder. The [ImageSize](#) property defines the size (width/height) of the icons within the control's Images collection. Use the [Replacelcon](#) method to add, remove or clear icons in the control's images collection. The tag can be used in the [Caption](#) property of the [Item](#) object. Also, the [Image](#) property assign an icon to the specified item.

property Ribbon.ImageSize as Long

Specifies the size to show the icons, check-boxes, radio-buttons, drop-down arrows and so on.

Type	Description
Long	A long expression that defines the size of icons the control displays

By default, the ImageSize property is 16 (pixels). The ImageSize property specifies the size of icons being loaded using the [Images](#) method. The control's Images collection is cleared if the ImageSize property is changed, so it is recommended to set the ImageSize property before calling the Images method. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. For instance, if the ICO file to load includes different types the one closest with the size specified by ImageSize property is loaded by Images method. The ImageSize property does NOT change the height for the control's font.

property Ribbon.Item (ID as Variant) as Item

Returns a specific Item object giving its identifier or caption.

Type	Description
ID as Variant	A Long expression that specifies the identifier of the item being requested or a String expression that specifies the caption of the item being requested.
Item	An Item object with associated identifier.

The Item property searches recursively the item with giving identifier/caption. The [ID](#) property of the Item object specifies the identifier of the item. The [Caption](#) property of the Item object specifies the caption of the item. The Item property gets the first Item object being found, if multiple objects with the same identifier are found, or Nothing, if no item with associated identifier is found. The [Item](#) property of the Items compared with the Item property of the eXRibbon is that the first look in the specified Items collection, while the second is looking for all Items in the menu object.

property Ribbon.ItemFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS) as Item

Retrieves the item from the cursor.

Type	Description
X as OLE_XPOS_PIXELS	A long expression that specifies the x-position to request item from
Y as OLE_YPOS_PIXELS	A long expression that specifies the y-position to request item from
Item	Returns the Item object from the cursor.

The ItemFromPoint(-1.-1) property determines the item from current cursor.

property Ribbon.Items as Items

Retrieves the control's Items collection.

Type	Description
Items	An Items object that holds a collection of Item objects.

The Items property gives access to the control's Items collection, so you can add, remove or update the items being shown in the ribbon control. The [Add](#) method adds a new item to the Items collection. The [ToString](#) property loads or saves the control items from a string. The [Remove](#) method removes a specified item. The [Item](#) property accesses an Item object giving its identifier or caption.

Is it possible to expand an item when it is clicked (tree,group,vertical)?

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items
    With .Add("Expand",2)
      .GroupPopup = 259 ' GroupPopupEnum.exGroupPopupVertical Or
GroupPopupEnum.exNoGroupPopupFrame Or
GroupPopupEnum.exGroupPopup
      .Check = True
      .ShowPopupOnChecked = True
    With .Items
      .Padding = "22,0,0,0"
      .Add("Radio 1").Radio = True
      .Add("Radio 2").Radio = True
      With .Add("Radio 3")
        .Radio = True
        .Checked = True
      End With
    End With
  End With
  .Checked = True
End With
End With
.Refresh
End With
```

VB6

With Ribbon1

With .Items

With .Add("Expand",2)

.GroupPopup = GroupPopupEnum.exGroupPopupVertical Or
GroupPopupEnum.exNoGroupPopupFrame Or GroupPopupEnum.exGroupPopup

.Check = True

.ShowPopupOnChecked = True

With .Items

.Padding = "22,0,0,0"

.Add("Radio 1").Radio = True

.Add("Radio 2").Radio = True

With .Add("Radio 3")

.Radio = True

.Checked = True

End With

End With

.Checked = True

End With

End With

.Refresh

End With

VB.NET

With Exribbon1

With .Items

With .Add("Expand",2)

.GroupPopup =

exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopupVertical Or
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup

.Check = True

.ShowPopupOnChecked = True

With .Items

.Padding = "22,0,0,0"

.Add("Radio 1").Radio = True

```

        .Add("Radio 2").Radio = True
    With .Add("Radio 3")
        .Radio = True
        .Checked = True
    End With
End With
.Checked = True
End With
End With
.Refresh()
End With

```

VB.NET for /COM

```

With AxRibbon1
    With .Items
        With .Add("Expand",2)
            .GroupPopup = EXRIBBONLib.GroupPopupEnum.exGroupPopupVertical Or
EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame Or
EXRIBBONLib.GroupPopupEnum.exGroupPopup
            .Check = True
            .ShowPopupOnChecked = True
        End With
    End With
    .Padding = "22,0,0,0"
    .Add("Radio 1").Radio = True
    .Add("Radio 2").Radio = True
    With .Add("Radio 3")
        .Radio = True
        .Checked = True
    End With
End With
.Checked = True
End With
.Refresh()
End With

```

```
/*
```

Copy and paste the following directives to your header file as it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control Library'

```
#import <ExRibbon.dll>
using namespace EXRIBBONLib;
```

```
*/
```

```
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Expand",long(2),vtMissing);
    var_Item-
> PutGroupPopup(EXRIBBONLib::GroupPopupEnum(EXRIBBONLib::exGroupPopupVert
| EXRIBBONLib::exNoGroupPopupFrame | EXRIBBONLib::exGroupPopup));
    var_Item->PutCheck(VARIANT_TRUE);
    var_Item->PutShowPopupOnChecked(VARIANT_TRUE);
    EXRIBBONLib::IItemsPtr var_Items1 = var_Item->GetItems();
    var_Items1->PutPadding(L"22,0,0,0");
    var_Items1->Add(L"Radio 1",vtMissing,vtMissing)->PutRadio(VARIANT_TRUE);
    var_Items1->Add(L"Radio 2",vtMissing,vtMissing)->PutRadio(VARIANT_TRUE);
    EXRIBBONLib::IItemPtr var_Item1 = var_Items1->Add(L"Radio
3",vtMissing,vtMissing);
    var_Item1->PutRadio(VARIANT_TRUE);
    var_Item1->PutChecked(VARIANT_TRUE);
    var_Item->PutChecked(VARIANT_TRUE);
spRibbon1->Refresh();
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    Exribbonlib_tlb::IItemPtr var_Item = var_Items-
>Add(L"Expand",TVariant(2),TNoParam());
    var_Item->GroupPopup =
Exribbonlib_tlb::GroupPopupEnum::exGroupPopupVertical |
Exribbonlib_tlb::GroupPopupEnum::exNoGroupPopupFrame |
```



```

Exribbonlib_tlb::GroupPopupEnum::exGroupPopup;
var_Item->Check = true;
var_Item->ShowPopupOnChecked = true;
Exribbonlib_tlb::IItemsPtr var_Items1 = var_Item->Items;
var_Items1->Padding = L"22,0,0,0";
var_Items1->Add(L"Radio 1",TNoParam(),TNoParam())->Radio = true;
var_Items1->Add(L"Radio 2",TNoParam(),TNoParam())->Radio = true;
Exribbonlib_tlb::IItemPtr var_Item1 = var_Items1->Add(L"Radio
3",TNoParam(),TNoParam());
var_Item1->Radio = true;
var_Item1->Checked = true;
var_Item->Checked = true;
Ribbon1->Refresh();

```

C#

```

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Expand",2,null);
var_Item.GroupPopup =
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopupVertical |
exontrol.EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
exontrol.EXRIBBONLib.GroupPopupEnum.exGroupPopup;
var_Item.Check = true;
var_Item.ShowPopupOnChecked = true;
exontrol.EXRIBBONLib.Items var_Items1 = var_Item.Items;
var_Items1.Padding = "22,0,0,0";
var_Items1.Add("Radio 1",null,null).Radio = true;
var_Items1.Add("Radio 2",null,null).Radio = true;
exontrol.EXRIBBONLib.Item var_Item1 = var_Items1.Add("Radio 3",null,null);
var_Item1.Radio = true;
var_Item1.Checked = true;
var_Item.Checked = true;
exribbon1.Refresh();

```

JScript/JavaScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var var_Item = var_Items.Add("Expand",2,null);
    var_Item.GroupPopup = 259;
    var_Item.Check = true;
    var_Item.ShowPopupOnChecked = true;
    var var_Items1 = var_Item.Items;
    var_Items1.Padding = "22,0,0,0";
    var_Items1.Add("Radio 1",null,null).Radio = true;
    var_Items1.Add("Radio 2",null,null).Radio = true;
    var var_Item1 = var_Items1.Add("Radio 3",null,null);
    var_Item1.Radio = true;
    var_Item1.Checked = true;
    var_Item.Checked = true;
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            With .Add("Expand",2)

```

```
.GroupPopup = 259 ' GroupPopupEnum.exGroupPopupVertical Or  
GroupPopupEnum.exNoGroupPopupFrame Or  
GroupPopupEnum.exGroupPopup
```

```
.Check = True
```

```
.ShowPopupOnChecked = True
```

```
With .Items
```

```
.Padding = "22,0,0,0"
```

```
.Add("Radio 1").Radio = True
```

```
.Add("Radio 2").Radio = True
```

```
With .Add("Radio 3")
```

```
.Radio = True
```

```
.Checked = True
```

```
End With
```

```
End With
```

```
.Checked = True
```

```
End With
```

```
End With
```

```
.Refresh
```

```
End With
```

```
End Function
```

```
</SCRIPT>
```

```
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;
```

```
EXRIBBONLib.Item var_Item = var_Items.Add("Expand",2,null);
```

```
var_Item.GroupPopup = EXRIBBONLib.GroupPopupEnum.exGroupPopupVertical  
| EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame |
```

```
EXRIBBONLib.GroupPopupEnum.exGroupPopup;
```

```
var_Item.Check = true;
```

```
var_Item.ShowPopupOnChecked = true;
```

```
EXRIBBONLib.Items var_Items1 = var_Item.Items;
```

```
var_Items1.Padding = "22,0,0,0";
```

```
var_Items1.Add("Radio 1",null,null).Radio = true;
```

```
var_Items1.Add("Radio 2",null,null).Radio = true;
```

```

EXRIBBONLib.Item var_Item1 = var_Items1.Add("Radio 3",null,null);
var_Item1.Radio = true;
var_Item1.Checked = true;
var_Item.Checked = true;
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    COM com_Item,com_Item1,com_Items,com_Items1;
    anytype var_Item,var_Item1,var_Items,var_Items1;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = com_Items.Add("Expand",COMVariant::createFromInt(2)); com_Item =
var_Item;
    com_Item.GroupPopup(259/*exGroupPopupVertical |
exNoGroupPopupFrame | exGroupPopup*/);
    com_Item.Check(true);
    com_Item.ShowPopupOnChecked(true);
    var_Items1 = com_Item.Items(); com_Items1 = var_Items1;
    com_Items1.Padding("22,0,0,0");
    var_Item1 = COM::createFromObject(com_Items1.Add("Radio 1"));
com_Item1 = var_Item1;
    com_Item1.Radio(true);
    var_Item1 = COM::createFromObject(com_Items1.Add("Radio 2"));
com_Item1 = var_Item1;
    com_Item1.Radio(true);
    var_Item1 = com_Items1.Add("Radio 3"); com_Item1 = var_Item1;
    com_Item1.Radio(true);
    com_Item1.Checked(true);
    com_Item.Checked(true);
    exribbon1.Refresh();

```

```
}
```

Delphi 8 (.NET only)

```
with AxRibbon1 do
begin
  with Items do
  begin
    with Add('Expand',TObject(2),Nil) do
    begin
      GroupPopup :=
Integer(EXRIBBONLib.GroupPopupEnum.exGroupPopupVertical) Or
Integer(EXRIBBONLib.GroupPopupEnum.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib.GroupPopupEnum.exGroupPopup);
      Check := True;
      ShowPopupOnChecked := True;
      with Items do
      begin
        Padding := '22,0,0,0';
        Add('Radio 1',Nil,Nil).Radio := True;
        Add('Radio 2',Nil,Nil).Radio := True;
        with Add('Radio 3',Nil,Nil) do
        begin
          Radio := True;
          Checked := True;
        end;
      end;
      Checked := True;
    end;
  end;
  Refresh();
end
```

Delphi (standard)

```
with Ribbon1 do
begin
  with Items do
```

```

begin
  with Add('Expand',OleVariant(2),Null) do
    begin
      GroupPopup := Integer(EXRIBBONLib_TLB.exGroupPopupVertical) Or
Integer(EXRIBBONLib_TLB.exNoGroupPopupFrame) Or
Integer(EXRIBBONLib_TLB.exGroupPopup);
      Check := True;
      ShowPopupOnChecked := True;
      with Items do
        begin
          Padding := '22,0,0,0';
          Add('Radio 1',Null,Null).Radio := True;
          Add('Radio 2',Null,Null).Radio := True;
          with Add('Radio 3',Null,Null) do
            begin
              Radio := True;
              Checked := True;
            end;
          end;
          Checked := True;
        end;
      end;
      Refresh();
    end
  end
end

```

VFP

```

with thisform.Ribbon1
  with .Items
    with .Add("Expand",2)
      .GroupPopup = 259 && GroupPopupEnum.exGroupPopupVertical Or
GroupPopupEnum.exNoGroupPopupFrame Or GroupPopupEnum.exGroupPopup
      .Check = .T.
      .ShowPopupOnChecked = .T.
    with .Items
      .Padding = "22,0,0,0"
      .Add("Radio 1").Radio = .T.
    end
  end
end

```

```

        .Add("Radio 2").Radio = .T.
    with .Add("Radio 3")
        .Radio = .T.
        .Checked = .T.
    endwhile
endwith
.Checked = .T.
endwith
.Refresh
endwith

```

dBASE Plus

```

local oRibbon,var_Item,var_Item1,var_Item2,var_Item3,var_Items,var_Items1

oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("Expand",2)
var_Item.GroupPopup = 259 /*exGroupPopupVertical | exNoGroupPopupFrame
| exGroupPopup*/
var_Item.Check = true
var_Item.ShowPopupOnChecked = true
var_Items1 = var_Item.Items
var_Items1.Padding = "22,0,0,0"
// var_Items1.Add("Radio 1").Radio = true
var_Item1 = var_Items1.Add("Radio 1")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.Radio = true]
endwith
// var_Items1.Add("Radio 2").Radio = true
var_Item2 = var_Items1.Add("Radio 2")
with (oRibbon)
    TemplateDef = [Dim var_Item2]
    TemplateDef = var_Item2

```

```

        Template = [var_Item2.Radio = true]
    endwhile
    var_Item3 = var_Items1.Add("Radio 3")
    var_Item3.Radio = true
    var_Item3.Checked = true
    var_Item.Checked = true
oRibbon.Refresh()

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Item1 as P
Dim var_Item2 as P
Dim var_Item3 as P
Dim var_Items as P
Dim var_Items1 as P

oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
    var_Item = var_Items.Add("Expand",2)
        var_Item.GroupPopup = 259 'exGroupPopupVertical +
exNoGroupPopupFrame + exGroupPopup
        var_Item.Check = .t.
        var_Item.ShowPopupOnChecked = .t.
    var_Items1 = var_Item.Items
        var_Items1.Padding = "22,0,0,0"
        'var_Items1.Add("Radio 1").Radio = .t.
    var_Item1 = var_Items1.Add("Radio 1")
    oRibbon.TemplateDef = "Dim var_Item1"
    oRibbon.TemplateDef = var_Item1
    oRibbon.Template = "var_Item1.Radio = True"

    'var_Items1.Add("Radio 2").Radio = .t.
    var_Item2 = var_Items1.Add("Radio 2")
    oRibbon.TemplateDef = "Dim var_Item2"

```



```
oRibbon.TemplateDef = var_Item2
oRibbon.Template = "var_Item2.Radio = True"

var_Item3 = var_Items1.Add("Radio 3")
var_Item3.Radio = .t.
var_Item3.Checked = .t.
var_Item.Checked = .t.
oRibbon.Refresh()
```

Visual Objects

```
local var_Item,var_Item1 as Item
local var_Items,var_Items1 as Items

var_Items := oDCOCX_Exontrol1:Items
var_Item := var_Items:Add("Expand",2,nil)
var_Item:GroupPopup := exGroupPopupVertical | exNoGroupPopupFrame |
exGroupPopup
var_Item:Check := true
var_Item:ShowPopupOnChecked := true
var_Items1 := var_Item:Items
var_Items1:Padding := "22,0,0,0"
var_Items1:Add("Radio 1",nil,nil):Radio := true
var_Items1:Add("Radio 2",nil,nil):Radio := true
var_Item1 := var_Items1:Add("Radio 3",nil,nil)
var_Item1:Radio := true
var_Item1.Checked := true
var_Item.Checked := true
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Item,var_Item1,var_Items,var_Items1

oRibbon = ole_1.Object
var_Items =oRibbon.Items
```

```

var_Item = var_Items.Add("Expand",2)
    var_Item.GroupPopup = 259 /*exGroupPopupVertical | exNoGroupPopupFrame
| exGroupPopup*/
    var_Item.Check = true
    var_Item.ShowPopupOnChecked = true
    var_Items1 = var_Item.Items
        var_Items1.Padding = "22,0,0,0"
        var_Items1.Add("Radio 1").Radio = true
        var_Items1.Add("Radio 2").Radio = true
        var_Item1 = var_Items1.Add("Radio 3")
            var_Item1.Radio = true
            var_Item1.Checked = true
        var_Item.Checked = true
oRibbon.Refresh()

```

Visual DataFlex

```

Procedure OnCreate
    Forward Send OnCreate
    Variant voltems
    Get ComItems to voltems
    Handle holtems
    Get Create (RefClass(cComItems)) to holtems
    Set pvComObject of holtems to voltems
        Variant voltem
        Get ComAdd of holtems "Expand" 2 Nothing to voltem
        Handle holtem
        Get Create (RefClass(cComItem)) to holtem
        Set pvComObject of holtem to voltem
            Set ComGroupPopup of holtem to (OLEexGroupPopupVertical +
OLEexNoGroupPopupFrame + OLEexGroupPopup)
            Set ComCheck of holtem to True
            Set ComShowPopupOnChecked of holtem to True
        Variant voltems1
        Get ComItems of holtem to voltems1
        Handle holtems1

```

```

Get Create (RefClass(cComItems)) to holtems1
Set pvComObject of holtems1 to voltems1
    Set ComPadding of holtems1 to "22,0,0,0"
Variant voltem1
Get ComAdd of holtems1 "Radio 1" Nothing Nothing to voltem1
Handle holtem1
Get Create (RefClass(cComItem)) to holtem1
Set pvComObject of holtem1 to voltem1
    Set ComRadio of holtem1 to True
Send Destroy to holtem1
Variant voltem2
Get ComAdd of holtems1 "Radio 2" Nothing Nothing to voltem2
Handle holtem2
Get Create (RefClass(cComItem)) to holtem2
Set pvComObject of holtem2 to voltem2
    Set ComRadio of holtem2 to True
Send Destroy to holtem2
Variant voltem3
Get ComAdd of holtems1 "Radio 3" Nothing Nothing to voltem3
Handle holtem3
Get Create (RefClass(cComItem)) to holtem3
Set pvComObject of holtem3 to voltem3
    Set ComRadio of holtem3 to True
    Set ComChecked of holtem3 to True
Send Destroy to holtem3
Send Destroy to holtems1
Set ComChecked of holtem to True
Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

```

PROCEDURE Main

LOCAL oForm

LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

LOCAL oltem, oltem1

LOCAL oltems, oltems1

LOCAL oRibbon

oForm := XbpDialog():new(AppDesktop())

oForm:drawingArea:clipChildren := .T.

oForm:create(,, {100,100}, {640,480},,, .F.)

oForm:close := {|| PostAppEvent(xbeP_Quit)}

oRibbon := XbpActiveXControl():new(oForm:drawingArea)

oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-CFBE431702E2}*/

oRibbon:create(,, {10,60},{610,370})

oltems := oRibbon:Items()

oltem := oltems:Add("Expand",2)

oltem:**GroupPopup** :=

259/*exGroupPopupVertical+exNoGroupPopupFrame+exGroupPopup*/

oltem:**Check** := .T.

oltem:**ShowPopupOnChecked** := .T.

oltems1 := oltem:Items()

oltems1:Padding := "22,0,0,0"

oltems1:Add("Radio 1"):Radio := .T.

oltems1:Add("Radio 2"):Radio := .T.

oltem1 := oltems1:Add("Radio 3")

oltem1:Radio := .T.

oltem1:Checked := .T.

oltem:Checked := .T.

oRibbon:Refresh()

oForm:Show()

DO WHILE nEvent != xbeP_Quit

nEvent := AppEvent(@mp1, @mp2, @oXbp)

oXbp:handleEvent(nEvent, mp1, mp2)

property Ribbon.LocalAppearance as AppearanceEnum

Retrieves or sets the local popup's appearance.

Type	Description
AppearanceEnum	<p>A AppearanceEnum expression that specifies the local's frame appearance, or a color expression whose last 7 bits in the high significant byte of the value indicates the index of the skin in the Appearance collection, being displayed as control's borders. For instance, if the Appearance = 0x1000000, indicates that the first skin object in the Appearance collection defines the control's border. <i>The Client object in the skin, defines the client area of the control. 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</i></p>

By default, the LocalAppearance property is -1. The visual appearance of the local popup is specified by the control's [Appearance](#) property, while the LocalAppearance property is -1. The [ShowLocalPopup](#) property specifies whether the item's popup is shown as local. Clicking any item inside a local popup makes the popup itself to close including all its descendent sub-menus, without closing any ascendant sub-menus. The [PopupAppearance](#) specifies a different visual appearance for the current submenu. When using EBN appearance, using the [PopupAppearance](#), LocalAppearance or [Appearance](#), the distance between margins/borders and items client area is indicated by the client object of the skin/ebn object.

The following screen shot shows the sub-menu with different appearances:



(single appearance)



(shadow appearance)



(ebn appearance)



(ebn appearance)

property Ribbon.Locked as Boolean

Locks or unlocks the control.

Type	Description
Boolean	A Boolean expression that specifies whether the control is locked or unlocked.

By default, the Locked property is False. Use the Locked property to lock the control. When the control is locked the inside elements look normal. Use the [Enabled](#) property of the Item object to disable a specific item. Use the [Enabled](#) property to enable or disable the control.

How can I lock the control, so no events occur, but no shown in gray as Enabled do?

VBA (MS Access, Excell...)

' SelectItem event - Occurs when the user selects the item.

```
Private Sub Ribbon1_SelectItem(ByVal Itm As Object)
```

```
    With Ribbon1
```

```
        Debug.Print( "SelectItem should not be fired while locked" )
```

```
    End With
```

```
End Sub
```

```
With Ribbon1
```

```
    .Locked = True
```

```
    With .Items
```

```
        .Add("Disabled").Enabled = False
```

```
        .Add "Item 2"
```

```
        .Add "Item 3"
```

```
    End With
```

```
    .Refresh
```

```
End With
```

VB6

' SelectItem event - Occurs when the user selects the item.

```
Private Sub Ribbon1_SelectItem(ByVal Itm As EXRIBBONLibCtl.Item)
```

```
    With Ribbon1
```

```
        Debug.Print( "SelectItem should not be fired while locked" )
```



```
End With
End Sub
```

```
With Ribbon1
```

```
  .Locked = True
```

```
  With .Items
```

```
    .Add("Disabled").Enabled = False
```

```
    .Add "Item 2"
```

```
    .Add "Item 3"
```

```
  End With
```

```
  .Refresh
```

```
End With
```

VB.NET

' **SelectItem event - Occurs when the user selects the item.**

```
Private Sub Exribbon1_SelectItem(ByVal sender As System.Object, ByVal Itm As  
exontrol.EXRIBBONLib.Item) Handles Exribbon1.SelectItem
```

```
  With Exribbon1
```

```
    Debug.Print( "SelectItem should not be fired while locked" )
```

```
  End With
```

```
End Sub
```

```
With Exribbon1
```

```
  .Locked = True
```

```
  With .Items
```

```
    .Add("Disabled").Enabled = False
```

```
    .Add("Item 2")
```

```
    .Add("Item 3")
```

```
  End With
```

```
  .Refresh()
```

```
End With
```

VB.NET for /COM

' **SelectItem event - Occurs when the user selects the item.**

```
Private Sub AxRibbon1_SelectItem(ByVal sender As System.Object, ByVal e As  
AxEXRIBBONLib._IRibbonEvents_SelectItemEvent) Handles AxRibbon1.SelectItem
```

```

With AxRibbon1
    Debug.Print( "SelectItem should not be fired while locked" )
End With
End Sub

```

```

With AxRibbon1
    .Locked = True
    With .Items
        .Add("Disabled").Enabled = False
        .Add("Item 2")
        .Add("Item 3")
    End With
    .Refresh()
End With

```

C++

// SelectItem event - Occurs when the user selects the item.

```

void OnSelectItemRibbon1(LPDISPATCH Itm)
{
    /*
        Copy and paste the following directives to your header file as
        it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control
        Library'
        #import <ExRibbon.dll>
        using namespace EXRIBBONLib;
    */
    EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
> GetControlUnknown();
    OutputDebugStringW( L"SelectItem should not be fired while locked" );
}

```

```

EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
> GetControlUnknown();
spRibbon1->PutLocked(VARIANT_TRUE);
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
var_Items->Add(L"Disabled",vtMissing,vtMissing)->PutEnabled(VARIANT_FALSE);

```

```
var_Items->Add(L"Item 2",vtMissing,vtMissing);
var_Items->Add(L"Item 3",vtMissing,vtMissing);
spRibbon1->Refresh();
```

C++ Builder

// SelectItem event - Occurs when the user selects the item.

```
void __fastcall TForm1::Ribbon1SelectItem(TObject *Sender,Exribbonlib_tlb::Item
*Itm)
{
    OutputDebugString( L"SelectItem should not be fired while locked" );
}
```

```
Ribbon1->Locked = true;
```

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
```

```
var_Items->Add(L"Disabled",TNoParam(),TNoParam())->Enabled = false;
```

```
var_Items->Add(L"Item 2",TNoParam(),TNoParam());
```

```
var_Items->Add(L"Item 3",TNoParam(),TNoParam());
```

```
Ribbon1->Refresh();
```

C#

// SelectItem event - Occurs when the user selects the item.

```
private void exribbon1_SelectItem(object sender,exontrol.EXRIBBONLib.Item Itm)
{
    System.Diagnostics.Debug.Print( "SelectItem should not be fired while locked" );
}
```

//this.exribbon1.SelectItem += new

exontrol.EXRIBBONLib.exg2antt.SelectItemEventHandler(this.exribbon1_SelectItem)

```
exribbon1.Locked = true;
```

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
```

```
var_Items.Add("Disabled",null,null).Enabled = false;
```

```
var_Items.Add("Item 2",null,null);
```

```
var_Items.Add("Item 3",null,null);
```

```
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload= 'Init()' >
<SCRIPT FOR= "Ribbon1" EVENT= "SelectItem(Itm)" LANGUAGE= "JScript" >
    alert( "SelectItem should not be fired while locked" );
</SCRIPT>

<OBJECT CLASSID= "clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id= "Ribbon1" > </OBJECT>

<SCRIPT LANGUAGE= "JScript" >
function Init()
{
    Ribbon1.Locked = true;
    var var_Items = Ribbon1.Items;
    var_Items.Add("Disabled",null,null).Enabled = false;
    var_Items.Add("Item 2",null,null);
    var_Items.Add("Item 3",null,null);
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload= 'Init()' >
<SCRIPT LANGUAGE= "VBScript" >
Function Ribbon1_SelectItem(Itm)
    With Ribbon1
        alert( "SelectItem should not be fired while locked" )
    End With
End Function
</SCRIPT>
```

```

<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Ribbon1
    .Locked = True
    With .Items
      .Add("Disabled").Enabled = False
      .Add "Item 2"
      .Add "Item 3"
    End With
    .Refresh
  End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

// SelectItem event - Occurs when the user selects the item.

```

private void axRibbon1_SelectItem(object sender,
AxEXRIBBONLib._IRibbonEvents_SelectItemEvent e)
{
  System.Diagnostics.Debug.Print( "SelectItem should not be fired while locked" );
}

//this.axRibbon1.SelectItem += new
AxEXRIBBONLib._IRibbonEvents_SelectItemEventHandler(this.axRibbon1_Select

axRibbon1.Locked = true;
EXRIBBONLib.Items var_Items = axRibbon1.Items;
var_Items.Add("Disabled",null,null).Enabled = false;
var_Items.Add("Item 2",null,null);
var_Items.Add("Item 3",null,null);
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

// SelectItem event - Occurs when the user selects the item.

```
void onEvent_SelectItem(COM _Itm)
{
    ;
    print( "SelectItem should not be fired while locked" );
}

public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

    super();

    exribbon1.Locked(true);
    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = COM::createFromObject(com_Items.Add("Disabled")); com_Item =
var_Item;
    com_Item.Enabled(false);
    com_Items.Add("Item 2");
    com_Items.Add("Item 3");
    exribbon1.Refresh();
}
```

Delphi 8 (.NET only)

// SelectItem event - Occurs when the user selects the item.

```
procedure TWinForm1.AxRibbon1_SelectItem(sender: System.Object; e:
AxEXRIBBONLib._IRibbonEvents_SelectItemEvent);
begin
    with AxRibbon1 do
    begin
        OutputDebugString( 'SelectItem should not be fired while locked' );
    end;
end;
```

```

    end
end;

with AxRibbon1 do
begin
    Locked := True;
    with Items do
    begin
        Add('Disabled',Nil,Nil).Enabled := False;
        Add('Item 2',Nil,Nil);
        Add('Item 3',Nil,Nil);
    end;
    Refresh();
end

```

Delphi (standard)

```

// SelectItem event - Occurs when the user selects the item.
procedure TForm1.Ribbon1SelectItem(ASender: TObject; Itm : IItem);
begin
    with Ribbon1 do
    begin
        OutputDebugString( 'SelectItem should not be fired while locked' );
    end
end;

with Ribbon1 do
begin
    Locked := True;
    with Items do
    begin
        Add('Disabled',Null,Null).Enabled := False;
        Add('Item 2',Null,Null);
        Add('Item 3',Null,Null);
    end;
    Refresh();
end

```

VFP

```
*** SelectItem event - Occurs when the user selects the item. ***
LPARAMETERS Itm
  with thisform.Ribbon1
    DEBUGOUT( "SelectItem should not be fired while locked" )
  endwith

with thisform.Ribbon1
  .Locked = .T.
  with .Items
    .Add("Disabled").Enabled = .F.
    .Add("Item 2")
    .Add("Item 3")
  endwith
  .Refresh
endwith
```

dBASE Plus

```
/*
with (this.ACTIVEX1.nativeObject)
  SelectItem = class::nativeObject_SelectItem
endwith
*/
// Occurs when the user selects the item.
function nativeObject_SelectItem(Itm)
  local oRibbon
  oRibbon = form.Activex1.nativeObject
  ? "SelectItem should not be fired while locked"
return

local oRibbon,var_Item,var_Items

oRibbon = form.Activex1.nativeObject
oRibbon.Locked = true
var_Items = oRibbon.Items
// var_Items.Add("Disabled").Enabled = false
```



```

var_Item = var_Items.Add("Disabled")
with (oRibbon)
    TemplateDef = [Dim var_Item]
    TemplateDef = var_Item
    Template = [var_Item.Enabled = false]
endwith
var_Items.Add("Item 2")
var_Items.Add("Item 3")
oRibbon.Refresh()

```

XBasic (Alpha Five)

' **Occurs when the user selects the item.**

```

function SelectItem as v (Itm as OLE::Exontrol.Ribbon.1::Item)
    Dim oRibbon as P
    oRibbon = topparent:CONTROL_ACTIVEX1.activex
    ? "SelectItem should not be fired while locked"
end function

```

```

Dim oRibbon as P
Dim var_Item as P
Dim var_Items as P

```

```

oRibbon = topparent:CONTROL_ACTIVEX1.activex
oRibbon.Locked = .t.
var_Items = oRibbon.Items

```

' **var_Items.Add("Disabled").Enabled = .f.**

```

var_Item = var_Items.Add("Disabled")
oRibbon.TemplateDef = "Dim var_Item"
oRibbon.TemplateDef = var_Item
oRibbon.Template = "var_Item.Enabled = False"

```

```

var_Items.Add("Item 2")
var_Items.Add("Item 3")
oRibbon.Refresh()

```

Visual Objects

```
METHOD OCX_Exontrol1SelectItem(Itm) CLASS MainDialog
    // SelectItem event - Occurs when the user selects the item.
    OutputDebugString(String2Psz( "SelectItem should not be fired while locked" ))
RETURN NIL

local var_Items as IItems

oDCOCX_Exontrol1:Locked := true
var_Items := oDCOCX_Exontrol1:Items
    var_Items:Add("Disabled",nil,nil):Enabled := false
    var_Items:Add("Item 2",nil,nil)
    var_Items:Add("Item 3",nil,nil)
oDCOCX_Exontrol1:Refresh()
```

PowerBuilder

```
/*begin event SelectItem(oleobject Itm) - Occurs when the user selects the item.*/
/*
    OleObject oRibbon
    oRibbon = ole_1.Object
    MessageBox("Information",string( "SelectItem should not be fired while locked" ))
*/
/*end event SelectItem*/

OleObject oRibbon,var_Items

oRibbon = ole_1.Object
oRibbon.Locked = true
var_Items = oRibbon.Items
    var_Items.Add("Disabled").Enabled = false
    var_Items.Add("Item 2")
    var_Items.Add("Item 3")
oRibbon.Refresh()
```

Visual DataFlex

// Occurs when the user selects the item.

Procedure OnComSelectItem Variant lltm

Forward Send OnComSelectItem lltm

Showln "SelectItem should not be fired while locked"

End_Procedure

Procedure OnCreate

Forward Send OnCreate

Set **ComLocked** to True

Variant voltems

Get ComItems to voltems

Handle holtems

Get Create (RefClass(cComItems)) to holtems

Set pvComObject of holtems to voltems

Variant voltem

Get ComAdd of holtems "Disabled" Nothing Nothing to voltem

Handle holtem

Get Create (RefClass(cComItem)) to holtem

Set pvComObject of holtem to voltem

Set ComEnabled of holtem to False

Send Destroy to holtem

Get ComAdd of holtems "Item 2" Nothing Nothing to Nothing

Get ComAdd of holtems "Item 3" Nothing Nothing to Nothing

Send Destroy to holtems

Send ComRefresh

End_Procedure

XBase++

PROCEDURE OnSelectItem(oRibbon,ltm)

DevOut("SelectItem should not be fired while locked")

RETURN

#include "AppEvent.ch"

#include "ActiveX.ch"

PROCEDURE Main

LOCAL oForm

LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

LOCAL oltems

LOCAL oRibbon

oForm := XbpDialog():new(AppDesktop())

oForm:drawingArea:clipChildren := .T.

oForm:create(„{100,100}, {640,480}„ .F.)

oForm:close := {|| PostAppEvent(xbeP_Quit)}

oRibbon := XbpActiveXControl():new(oForm:drawingArea)

oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-CFBE431702E2}*/

oRibbon:create(„ {10,60},{610,370})

oRibbon:SelectItem := {||Itm| OnSelectItem(oRibbon,Itm)} /*Occurs when the user selects the item.*/

oRibbon:**Locked** := .T.

oltems := oRibbon:Items()

oltems:Add("Disabled"):Enabled := .F.

oltems:Add("Item 2")

oltems:Add("Item 3")

oRibbon:Refresh()

oForm:Show()

DO WHILE nEvent != xbeP_Quit

nEvent := AppEvent(@mp1, @mp2, @oXbp)

oXbp:handleEvent(nEvent, mp1, mp2)

ENDDO

RETURN

property Ribbon.Notifier as Long

Retrieves or sets the handle of the window that receives notifications/WM_COMMAND messages.

Type	Description
Long	A Long expression that specifies the handle of the window that receives the WM_COMMAND when the user selects, check/uncheck, edit an item.

The trial/evaluation version of the control limits firing the event/WM_COMMAND. In other words, using the trial/evaluation version won't fire the event/WM_COMMAND every time.

By default, the Notifier property is 0, which indicates that the property has no effect. Set the Notifier property to a window that you want to receive notification of the control through the WM_COMMAND message. For instance, in VFP or C++ it would be easier to handle the events of the control using the WM_COMMAND messages, rather than using sink interfaces.

The wParam parameter of the WM_COMMAND message carries the identifier of the event which occurred like listed below:

- **0** (exSelectItem), occurs when the user selects/clicks an item
- **1** (exCheckItem), occurs when the user clicks the item's check box, or check the item's checkbox
- **2** (exUncheckItem), occurs when the user clicks the item's check box, or uncheck the item's checkbox
- **3** (exEditChangeItem), occurs when the content of the item's editor is changed.

The lParam parameter of the WM_COMMAND message carries the identifier of the item who fired the event. You can use the [Item](#) property to access the control's item giving its identifier. The [ID](#) property specifies the item's identifier.

In VFP, you have to assign the hWnd property of the form to the Notifier property of the control as follows:

```
ribbon.Notifier = thisform.HWnd
```

while the following code:

```
BINDEVENT( thisform.HWnd, 273, thisform, "oncommand" )
```

adds a handler oncommand for the WM_COMMAND message (273 or 0x111 in hexa, is

the identifier of the WM_COMMAND message).

The oncommand may look like:

```
LPARAMETERS hWnd, uMsg, wParam, lParam
```

```
?wParam
```

```
*CheckItem
```

```
IF ( wParam = 1 ) then
```

```
    thisform.ribbon_CheckItem(lParam)
```

```
ELSE
```

```
    * UncheckItem
```

```
    IF ( wParam = 2 ) then
```

```
        thisform.ribbon_UncheckItem(lParam)
```

```
    ENDIF
```

```
ENDIF
```

property Ribbon.Picture as IPictureDisp

Retrieves or sets a graphic to be displayed in the control.

Type	Description
IPictureDisp	A Picture object that indicates the control's picture.

By default, the control has no picture associated. The control uses the [PictureDisplay](#) property to determine how the picture is displayed on the control's background. Use the Picture property to display a picture on the control's background.

property Ribbon.PictureDisplay as PictureDisplayEnum

Retrieves or sets a value that indicates the way how the graphic is displayed on the control's background

Type	Description
PictureDisplayEnum	A PictureDisplayEnum expression that indicates the way how the control's picture is displayed.

By default, the PictureDisplay property is exTile. The PictureDisplay property specifies how the [Picture](#) is displayed on the control's background. If the control has no picture associated the PictureDisplay property has no effect.

property Ribbon.PopupAppearance as AppearanceEnum

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

Type	Description
AppearanceEnum	A AppearanceEnum expression that specifies menu's appearance.

By default, the PopupAppearance property is NoBorder. The [Background\(exMenuFlatLineColor\)](#) property indicates the color of line that divides the left to right side of the menu, when the PopupFlatAppearance property is True. The [PopupFlatBackColor](#) property indicates the color to show the left part of the menu, when the PopupFlatAppearance property is True. The [BackColor](#) property specifies the menu's background color. The [Background\(exMenuSeparatorItem\)](#) property specifies the visual appearance of the separator items.

property Ribbon.PopupFlatAppearance as Boolean

Specifies whether the control shows a flat appearance for sub-menus.

Type	Description
Boolean	A Boolean expression that specifies whether the popup menu displays the flay/gray portion to the left.

By default, the PopupFlatAppearance property is True. Use the [Popup FlatBackColor](#) property to specify the background color of the left side of the popup. Use the [PopupFlatImageWidth](#) property to specify the width of the column that displays icons/images/check or radio buttons.

property Ribbon.PopupFlatBackColor as Color

Specifies the color to left part of the menu.

Type	Description
Color	A Color expression that indicates the control's 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 PopupFlatBackColor property to specify the background color of the left side of the popup. This property has effect while the control's [PopupFlatAppearance](#) property is True. The [BackColor](#) property specifies the control's background color. The [ForeColor](#) property specifies the control's foreground color. The [SelBackColor](#) property specifies the visual appearance of the item being selected / highlighted. The [SelForeColor](#) property specifies the foreground color of the item being selected / highlighted. The [Background](#) property specifies the visual appearance for different parts of the control. The [Appearance](#) property specifies the menu's frame appearance. The [Background\(exMenuFlatLineColor\)](#) property indicates the color of line that divides the left to right side of the popup menu.

property Ribbon.PopupFlatImageWidth as Long

Specifies the width of the column to display the icons/images when the control's PopupFlatAppearance is True.

Type	Description
Long	A Long expression that specifies the width of the column that displays icons/images/check or radio buttons, when the control's PopupFlatAppearance is True.

By default, the PopupFlatImageWidth property is 16 pixels wide. Use the PopupFlatImageWidth property to specify the width of the column that displays icons/images/check or radio buttons. The [Image](#) / [HTMLImage](#) property assigns an icon / picture to the item. The tag can be used in the [Caption](#) property of the [Item](#) object to display an Icon or a custom-size picture.

property Ribbon.PopupIncrementalSearch as IncrementalSearchEnum

Specifies how the control searches for the objects while user types characters.

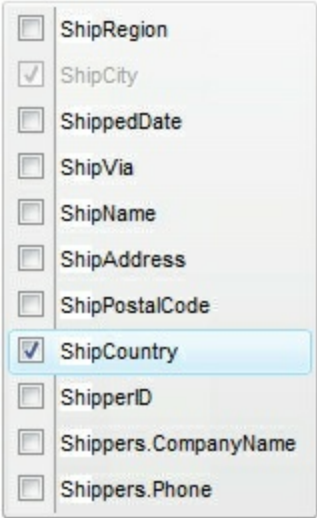
Type	Description
IncrementalSearchEnum	An IncrementalSearchEnum expression that specifies the type of incremental searching the control performs once the user types characters on the context menu.

"In computing, incremental search, incremental find or real-time suggestions is a user interface interaction method to progressively search for and filter through text. As the user types text, one or more possible matches for the text are found and immediately presented to the user. " By default, the PopupIncrementalSearch property is exlSearchStartWith + exlSearchFilterFor, in other words, the control filter for items that match the typing characters. Use the PopupIncrementalSearch property on exNoIncrementalSearch to disable/prevent the incremental searching in your context menu. While the incremental search is on, the F3 or Shift + F3, finds the next occurrence or previously occurrence. The Back key deletes the last character of the incremental search string, while the Ctrl + Back key removes the entire incremental search string. If the PopupIncrementalSearch property is exNoIncrementalSearch, you can use the item's [Shortcut](#) property to define the key combination that the user can press to select the item quickly.

You can use the PopupIncrementalSearch property:

- to highlight the items that match the typing characters
- to display just the items that match the typing characters

The following screen shot shows the control with IncrementalSearch property on exlSearchStartWith + exlSearchFilterFor, while the user types **"shi"**:



he following screen shot shows the control with IncrementalSearch property on exlSearchStartWith, while the user types **"shi"**:

▲

☐

ShipAddress

☐

ShipPostalCode

☐

ShipCountry

☐

ShipperID

☐

Shippers.CompanyName

☐

Shippers.Phone

☐

SupplierID

☐

Suppliers.CompanyName

☐

ContactName

☐

ContactTitle

☐

Address

☐

City

▼

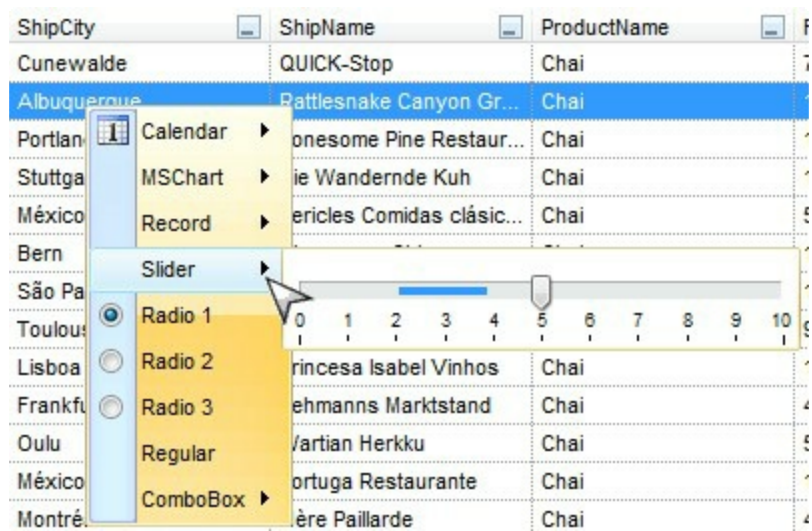
property Ribbon.PopupVisibility as Long

Specify the popup's visibility in percents: 90% is barely visible, and 10% is nearly opaque.

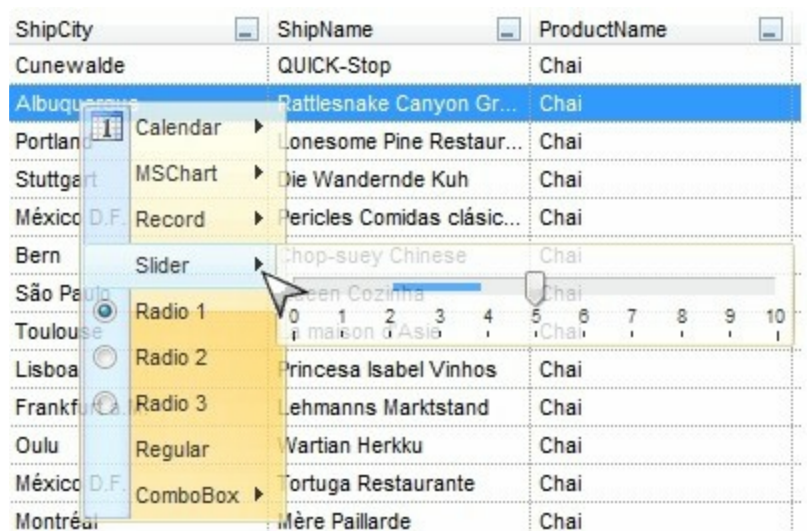
Type	Description
Long	A long expression that indicates the visibility of the popup menus.

By default, the Visibility is 100. Use the Visibility property to change the menu's visibility.

The following screen shot shows the menu when the Visibility is 100 (opaque, by default):



The following screen shot shows the menu when the Visibility is 80 (semi-transparent):



method Ribbon.Refresh ()

Refreshes the control.

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

Call the Refresh method to refresh the control's content, which includes resizing/visibility of all including elements. Use the [Update](#) method to validate just the drawing area of the ribbon control. For instance, if you are changing the item's [Caption](#), if an [OleEvent](#) occurs.

method Ribbon.Replacelcon ([Icon as Variant], [Index as Variant])

Adds a new icon, replaces an icon or clears the control's image list.

Type	Description
Icon as Variant	A long expression that indicates the icon's handle.
Index as Variant	A long expression that indicates the index where icon is inserted.

Return	Description
Long	A long expression that indicates the index of the icon in the images collection

Use the Replacelcon property to add, remove or replace an icon in the control's images collection. Also, the Replacelcon property can clear the images collection. Use the [Images](#) method to attach a image list to the control. The tag can be used in the [Caption](#) property of the [Item](#) object. Also, the [Image](#) property assign an icon to the specified item.

The following VB sample adds a new icon to control's images list:

```
i = Ribbon1.Replacelcon( LoadPicture("d:\icons\help.ico").Handle), i specifies the index where the icon is added
```

The following VB sample replaces an icon into control's images list::

```
i = Ribbon1.Replacelcon( LoadPicture("d:\icons\help.ico").Handle, 0), i is zero, so the first icon is replaced.
```

The following VB sample removes an icon from control's images list:

```
ExRibbon1.Replacelcon 0, i, i specifies the index of icon removed.
```

The following VB clears the control's icons collection:

```
ExRibbon1.Replacelcon 0, -1
```

property Ribbon.RequiredHeight as Long

Indicates the height to fit all elements within the control.

Type	Description
Long	A long expression that specifies the height to fit all elements within the control.

The RequiredHeight property gets the height (in pixels) required by the control all all visible items fit the control's client area. The [RequiredWidth](#) property gets the width (in pixels) required by the control all all visible items fit the control's client area.

The following samples show how you can get the required size, so all elements of the control fits its client area

VBA (MS Access, Excell...)

```
With Ribbon1
    With .Items
        .Add "Item 1"
        .Add "Item 2"
    End With
    .Refresh
    Debug.Print( .RequiredWidth )
    Debug.Print( .RequiredHeight )
End With
```

VB6

```
With Ribbon1
    With .Items
        .Add "Item 1"
        .Add "Item 2"
    End With
    .Refresh
    Debug.Print( .RequiredWidth )
    Debug.Print( .RequiredHeight )
End With
```

VB.NET

```
With Exribbon1
```

```
With .Items
```

```
.Add("Item 1")
```

```
.Add("Item 2")
```

```
End With
```

```
.Refresh()
```

```
Debug.Print( .RequiredWidth )
```

```
Debug.Print( .RequiredHeight )
```

```
End With
```

VB.NET for /COM

```
With AxRibbon1
```

```
With .Items
```

```
.Add("Item 1")
```

```
.Add("Item 2")
```

```
End With
```

```
.Refresh()
```

```
Debug.Print( .RequiredWidth )
```

```
Debug.Print( .RequiredHeight )
```

```
End With
```

C++

```
/*
```

```
Copy and paste the following directives to your header file as  
it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control  
Library'
```

```
#import <ExRibbon.dll>
```

```
using namespace EXRIBBONLib;
```

```
*/
```

```
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
```

```
>GetControlUnknown();
```

```
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
```

```
var_Items->Add(L"Item 1",vtMissing,vtMissing);
```

```
var_Items->Add(L"Item 2",vtMissing,vtMissing);
```

```
spRibbon1->Refresh();
```

```
OutputDebugStringW( _bstr_t(spRibbon1->GetRequiredWidth()) );  
OutputDebugStringW( _bstr_t(spRibbon1->GetRequiredHeight()) );
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;  
    var_Items->Add(L"Item 1",TNoParam(),TNoParam());  
    var_Items->Add(L"Item 2",TNoParam(),TNoParam());  
Ribbon1->Refresh();  
OutputDebugString( PChar(Ribbon1->RequiredWidth) );  
OutputDebugString( PChar(Ribbon1->RequiredHeight) );
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
    var_Items.Add("Item 1",null,null);  
    var_Items.Add("Item 2",null,null);  
exribbon1.Refresh();  
System.Diagnostics.Debug.Print( exribbon1.RequiredWidth.ToString() );  
System.Diagnostics.Debug.Print( exribbon1.RequiredHeight.ToString() );
```

JScript/JavaScript

```
<BODY onload='Init()>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    var var_Items = Ribbon1.Items;  
        var_Items.Add("Item 1",null,null);  
        var_Items.Add("Item 2",null,null);  
    Ribbon1.Refresh();  
    alert( Ribbon1.RequiredWidth );  
}
```

```
    alert( Ribbon1.RequiredHeight );  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
    With Ribbon1  
        With .Items  
            .Add "Item 1"  
            .Add "Item 2"  
        End With  
        .Refresh  
        alert( .RequiredWidth )  
        alert( .RequiredHeight )  
    End With  
End Function  
</SCRIPT>  
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;  
var_Items.Add("Item 1",null,null);  
var_Items.Add("Item 2",null,null);  
axRibbon1.Refresh();  
System.Diagnostics.Debug.Print( axRibbon1.RequiredWidth.ToString() );  
System.Diagnostics.Debug.Print( axRibbon1.RequiredHeight.ToString() );
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Items;
    anytype var_Items;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    com_Items.Add("Item 1");
    com_Items.Add("Item 2");
    exribbon1.Refresh();
    print( exribbon1.RequiredWidth() );
    print( exribbon1.RequiredHeight() );
}
```

Delphi 8 (.NET only)

```
with AxRibbon1 do
begin
    with Items do
    begin
        Add('Item 1',Nil,Nil);
        Add('Item 2',Nil,Nil);
    end;
    Refresh();
    OutputDebugString( RequiredWidth );
    OutputDebugString( RequiredHeight );
end
```

Delphi (standard)

```
with Ribbon1 do
begin
    with Items do
    begin
```

```

    Add('Item 1',Null,Null);
    Add('Item 2',Null,Null);
end;
Refresh();
OutputDebugString( RequiredWidth );
OutputDebugString( RequiredHeight );
end

```

VFP

```

with thisform.Ribbon1
  with .Items
    .Add("Item 1")
    .Add("Item 2")
  endwith
.Refresh
DEBUGOUT( .RequiredWidth )
DEBUGOUT( .RequiredHeight )
endwith

```

dBASE Plus

```

local oRibbon,var_Items

oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
  var_Items.Add("Item 1")
  var_Items.Add("Item 2")
oRibbon.Refresh()
? Str(oRibbon.RequiredWidth)
? Str(oRibbon.RequiredHeight)

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Items as P

```

```
oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
    var_Items.Add("Item 1")
    var_Items.Add("Item 2")
oRibbon.Refresh()
? oRibbon.RequiredWidth
? oRibbon.RequiredHeight
```

Visual Objects

```
local var_Items as IItems

var_Items := oDCOCX_Exontrol1.Items
    var_Items.Add("Item 1",nil,nil)
    var_Items.Add("Item 2",nil,nil)
oDCOCX_Exontrol1.Refresh()
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1.RequiredWidth) ))
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1.RequiredHeight) ))
```

PowerBuilder

```
OleObject oRibbon,var_Items

oRibbon = ole_1.Object
var_Items = oRibbon.Items
    var_Items.Add("Item 1")
    var_Items.Add("Item 2")
oRibbon.Refresh()
MessageBox("Information",string( String(oRibbon.RequiredWidth) ))
MessageBox("Information",string( String(oRibbon.RequiredHeight) ))
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
```



```

Variant voltems
Get ComItems to voltems
Handle holtems
Get Create (RefClass(cComItems)) to holtems
Set pvComObject of holtems to voltems
    Get ComAdd of holtems "Item 1" Nothing Nothing to Nothing
    Get ComAdd of holtems "Item 2" Nothing Nothing to Nothing
Send Destroy to holtems
Send ComRefresh
ShowIn (ComRequiredWidth(Self))
ShowIn (ComRequiredHeight(Self))
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oltems := oRibbon:Items()
    oltems:Add("Item 1")
    oltems:Add("Item 2")

```

```
oRibbon:Refresh()
```

```
DevOut( Transform(oRibbon:RequiredWidth(), "" )
```

```
DevOut( Transform(oRibbon:RequiredHeight(), "" )
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Ribbon.RequiredWidth as Long

Indicates the width to fit all elements within the control.

Type	Description
Long	A long expression that specifies the width to fit all elements within the control.

The RequiredWidth property gets the width (in pixels) required by the control all all visible items fit the control's client area. The [RequiredHeight](#) property gets the height (in pixels) required by the control all all visible items fit the control's client area.

The following samples show how you can get the required size, so all elements of the control fits its client area

VBA (MS Access, Excell...)

```
With Ribbon1
    With .Items
        .Add "Item 1"
        .Add "Item 2"
    End With
    .Refresh
    Debug.Print( .RequiredWidth )
    Debug.Print( .RequiredHeight )
End With
```

VB6

```
With Ribbon1
    With .Items
        .Add "Item 1"
        .Add "Item 2"
    End With
    .Refresh
    Debug.Print( .RequiredWidth )
    Debug.Print( .RequiredHeight )
End With
```

VB.NET

```
With Exribbon1
```

```
With .Items
```

```
.Add("Item 1")
```

```
.Add("Item 2")
```

```
End With
```

```
.Refresh()
```

```
Debug.Print( .RequiredWidth )
```

```
Debug.Print( .RequiredHeight )
```

```
End With
```

VB.NET for /COM

```
With AxRibbon1
```

```
With .Items
```

```
.Add("Item 1")
```

```
.Add("Item 2")
```

```
End With
```

```
.Refresh()
```

```
Debug.Print( .RequiredWidth )
```

```
Debug.Print( .RequiredHeight )
```

```
End With
```

C++

```
/*
```

```
Copy and paste the following directives to your header file as  
it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control  
Library'
```

```
#import <ExRibbon.dll>
```

```
using namespace EXRIBBONLib;
```

```
*/
```

```
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
```

```
>GetControlUnknown();
```

```
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
```

```
var_Items->Add(L"Item 1",vtMissing,vtMissing);
```

```
var_Items->Add(L"Item 2",vtMissing,vtMissing);
```

```
spRibbon1->Refresh();
```

```
OutputDebugStringW( _bstr_t(spRibbon1->GetRequiredWidth()) );  
OutputDebugStringW( _bstr_t(spRibbon1->GetRequiredHeight()) );
```

C++ Builder

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;  
    var_Items->Add(L"Item 1",TNoParam(),TNoParam());  
    var_Items->Add(L"Item 2",TNoParam(),TNoParam());  
Ribbon1->Refresh();  
OutputDebugString( PChar(Ribbon1->RequiredWidth) );  
OutputDebugString( PChar(Ribbon1->RequiredHeight) );
```

C#

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
    var_Items.Add("Item 1",null,null);  
    var_Items.Add("Item 2",null,null);  
exribbon1.Refresh();  
System.Diagnostics.Debug.Print( exribbon1.RequiredWidth.ToString() );  
System.Diagnostics.Debug.Print( exribbon1.RequiredHeight.ToString() );
```

JScript/JavaScript

```
<BODY onload='Init()>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    var var_Items = Ribbon1.Items;  
        var_Items.Add("Item 1",null,null);  
        var_Items.Add("Item 2",null,null);  
    Ribbon1.Refresh();  
    alert( Ribbon1.RequiredWidth );  
}
```

```
    alert( Ribbon1.RequiredHeight );  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
    With Ribbon1  
        With .Items  
            .Add "Item 1"  
            .Add "Item 2"  
        End With  
        .Refresh  
        alert( .RequiredWidth )  
        alert( .RequiredHeight )  
    End With  
End Function  
</SCRIPT>  
</BODY>
```

C# for /COM

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;  
var_Items.Add("Item 1",null,null);  
var_Items.Add("Item 2",null,null);  
axRibbon1.Refresh();  
System.Diagnostics.Debug.Print( axRibbon1.RequiredWidth.ToString() );  
System.Diagnostics.Debug.Print( axRibbon1.RequiredHeight.ToString() );
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Items;
    anytype var_Items;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    com_Items.Add("Item 1");
    com_Items.Add("Item 2");
    exribbon1.Refresh();
    print( exribbon1.RequiredWidth() );
    print( exribbon1.RequiredHeight() );
}
```

Delphi 8 (.NET only)

```
with AxRibbon1 do
begin
    with Items do
    begin
        Add('Item 1',Nil,Nil);
        Add('Item 2',Nil,Nil);
    end;
    Refresh();
    OutputDebugString( RequiredWidth );
    OutputDebugString( RequiredHeight );
end
```

Delphi (standard)

```
with Ribbon1 do
begin
    with Items do
    begin
```

```

    Add('Item 1',Null,Null);
    Add('Item 2',Null,Null);
end;
Refresh();
OutputDebugString( RequiredWidth );
OutputDebugString( RequiredHeight );
end

```

VFP

```

with thisform.Ribbon1
  with .Items
    .Add("Item 1")
    .Add("Item 2")
  endwith
.Refresh
DEBUGOUT( .RequiredWidth )
DEBUGOUT( .RequiredHeight )
endwith

```

dBASE Plus

```

local oRibbon,var_Items

oRibbon = form.ActiveX1.nativeObject
var_Items = oRibbon.Items
  var_Items.Add("Item 1")
  var_Items.Add("Item 2")
oRibbon.Refresh()
? Str(oRibbon.RequiredWidth)
? Str(oRibbon.RequiredHeight)

```

XBasic (Alpha Five)

```

Dim oRibbon as P
Dim var_Items as P

```



```
oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
    var_Items.Add("Item 1")
    var_Items.Add("Item 2")
oRibbon.Refresh()
? oRibbon.RequiredWidth
? oRibbon.RequiredHeight
```

Visual Objects

```
local var_Items as IItems

var_Items := oDCOCX_Exontrol1.Items
    var_Items.Add("Item 1",nil,nil)
    var_Items.Add("Item 2",nil,nil)
oDCOCX_Exontrol1.Refresh()
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1.RequiredWidth) ))
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1.RequiredHeight) ))
```

PowerBuilder

```
OleObject oRibbon,var_Items

oRibbon = ole_1.Object
var_Items = oRibbon.Items
    var_Items.Add("Item 1")
    var_Items.Add("Item 2")
oRibbon.Refresh()
MessageBox("Information",string( String(oRibbon.RequiredWidth) ))
MessageBox("Information",string( String(oRibbon.RequiredHeight) ))
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
```

```

Variant voltems
Get ComItems to voltems
Handle holtems
Get Create (RefClass(cComItems)) to holtems
Set pvComObject of holtems to voltems
    Get ComAdd of holtems "Item 1" Nothing Nothing to Nothing
    Get ComAdd of holtems "Item 2" Nothing Nothing to Nothing
Send Destroy to holtems
Send ComRefresh
ShowIn (ComRequiredWidth(Self))
ShowIn (ComRequiredHeight(Self))
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( ,, {100,100}, {640,480},,, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(,, {10,60},{610,370} )

    oltems := oRibbon:Items()
    oltems:Add("Item 1")
    oltems:Add("Item 2")

```

```
oRibbon:Refresh()
```

```
DevOut( Transform(oRibbon:RequiredWidth(), "" )
```

```
DevOut( Transform(oRibbon:RequiredHeight(), "" )
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Ribbon.SelBackColor as Color

Retrieves or sets a value that indicates the selection background color.

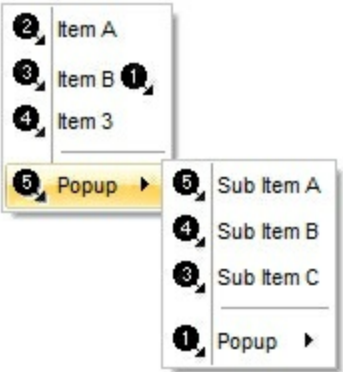
Type	Description
Color	A Color expression that specifies the background color / visual appearance of the selected item. The last 7 bits in the high significant byte of the color indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove 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.

The SelBackColor property specifies the visual appearance of the item being selected / highlighted. The [SelForeColor](#) property specifies the foreground color of the item being selected / highlighted. The [ForeColor](#) property specifies the control's foreground color. The [BackColor](#) property specifies the control's background color. The [Background](#) property specifies the visual appearance for different parts of the control. The [Appearance](#) property specifies the menu's frame appearance.

The following screen shot shows the control's selection with the default colors:



The following screen shot shows the control's selection with a different visual appearance:



property Ribbon.SelForeColor as Color

Retrieves or sets a value that indicates the selection foreground color.

Type	Description
Color	A Color expression that specifies the foreground color to show the selected / highlighted item.

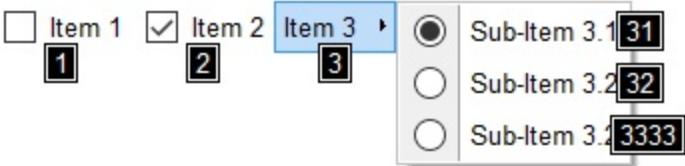
Currently, the SelForeColor property is super-classed by the [Background\(exMenuHotForeColor\)](#) property. The [SelBackColor](#) property specifies the visual appearance of the item being selected / highlighted. The [ForeColor](#) property specifies the control's foreground color. The [BackColor](#) property specifies the control's background color. The [Background](#) property specifies the visual appearance for different parts of the control. The [Appearance](#) property specifies the menu's frame appearance.

property Ribbon.ShortcutKeyAlignH as ShortcutKeyAlignEnum

Defines the alignment of the UI shortcut keys relative to the item that displays it, when the holder arranges the items horizontally.

Type	Description
ShortcutKeyAlignEnum	A ShortcutKeyAlignEnum expression that defines the alignment of the UI shortcut keys relative to the item that displays it, when the holder arranges the items horizontally.

The `ShortcutKeyAlignH` / [ShortcutKeyAlignV](#) defines the alignment of the UI shortcut keys relative to the item that displays it, when the holder arranges the items horizontally / vertically. The [ShortcutKeyVisible](#) property indicates the way the control shows the available keys when user presses any modifier keys such as SHIFT, CTRL or ALT. The [Shortcut](#) property specifies the key combination that the user can press to select the item quickly. The [ShortcutKeysInfo](#) property returns the list of shortcut keys that are currently available.



The following properties customize the format/look the shortcut keys are displayed:

- The [ShortcutKeyFormat](#) property specifies the expression that defines the format (including HTML tags) to display the shortcut keys.
- [Background\(exShortcutKeyAppearance\)](#) / [Background\(exShortcutKeyBackColor\)](#) / [Background\(exShortcutKeyForeColor\)](#) specifies the visual appearance of the shortcut's background / foreground
- The [ShortcutKeyTransparent](#) property Indicates the percent of transparency the shortcut keys are being displayed with.
- The [ShortcutKeyPadding](#) property specifies the UI shortcut key's content padding.
- The [ShortcutKeyExtPaddingH](#) / [ShortcutKeyExtPaddingV](#) specifies the padding to be applied to the item (arranged horizontally/vertically) that displays the UI shortcut, so the `ShortcutKeyAlign` property aligns the UI shortcut relative to this.

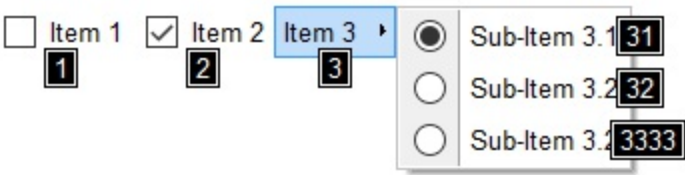
The [ShortcutKeyPressedModifiers](#) property indicates the combination of modifier keys whose shortcut keys are currently visible.

property Ribbon.ShortcutKeyAlignV as ShortcutKeyAlignEnum

Defines the alignment of the UI shortcut keys relative to the item that displays it, when the holder arranges the items vertically.

Type	Description
ShortcutKeyAlignEnum	A ShortcutKeyAlignEnum expression that defines the alignment of the UI shortcut keys relative to the item that displays it, when the holder arranges the items vertically.

The [ShortcutKeyAlignH](#) / [ShortcutKeyAlignV](#) defines the alignment of the UI shortcut keys relative to the item that displays it, when the holder arranges the items horizontally / vertically. The [ShortcutKeyVisible](#) property indicates the way the control shows the available keys when user presses any modifier keys such as SHIFT, CTRL or ALT. The [Shortcut](#) property specifies the key combination that the user can press to select the item quickly. The [ShortcutKeysInfo](#) property returns the list of shortcut keys that are currently available.



- The following properties customize the format/look the shortcut keys are displayed:
- The [ShortcutKeyFormat](#) property specifies the expression that defines the format (including HTML tags) to display the shortcut keys.
 - [Background\(exShortcutKeyAppearance\)](#) / [Background\(exShortcutKeyBackColor\)](#) / [Background\(exShortcutKeyForeColor\)](#) specifies the visual appearance of the shortcut's background / foreground
 - The [ShortcutKeyTransparent](#) property Indicates the percent of transparency the shortcut keys are being displayed with.
 - The [ShortcutKeyPadding](#) property specifies the UI shortcut key's content padding.
 - The [ShortcutKeyExtPaddingH](#) / [ShortcutKeyExtPaddingV](#) specifies the padding to be applied to the item (arranged horizontally/vertically) that displays the UI shortcut, so the [ShortcutKeyAlign](#) property aligns the UI shortcut relative to this.

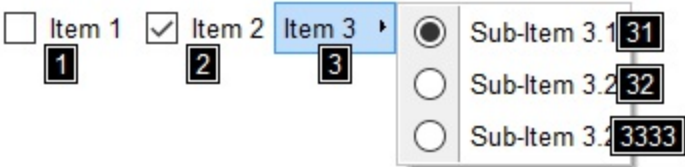
The [ShortcutKeyPressedModifiers](#) property indicates the combination of modifier keys whose shortcut keys are currently visible.

property Ribbon.ShortcutKeyExtPaddingH as String

Specifies the padding to be applied to the item (arranged horizontally) that displays the UI shortcut, so the ShortcutKeyAlign property aligns the UI shortcut relative to this.

Type	Description
String	A String expression that specifies the padding to be applied to the item (arranged horizontally) that displays the UI shortcut, so the ShortcutKeyAlign property aligns the UI shortcut relative to this. The list is separated by comma character and indicates left, top, right and bottom exterior padding.

By default, the ShortcutKeyExtPaddingH property is "width/4, height/4 , width/4 ,height/4". The ShortcutKeyExtPaddingH / [ShortcutKeyExtPaddingV](#) specifies the padding to be applied to the item (arranged horizontally/vertically) that displays the UI shortcut, so the ShortcutKeyAlign property aligns the UI shortcut relative to this. The [ShortcutKeyVisible](#) property indicates the way the control shows the available keys when user presses any modifier keys such as SHIFT, CTRL or ALT. The [Shortcut](#) property specifies the key combination that the user can press to select the item quickly. The [ShortcutKeysInfo](#) property returns the list of shortcut keys that are currently available.



For instance:

- "4,4,4,4" indicates a fixed exterior padding.
- "width/4, height/4 , width/4 ,height/4" indicates a relative exterior padding.

The ShortcutKeyExtPaddingH property supports the following predefined keywords:

- **width** keyword, returns the width required to display the shortcut's caption
- **height** keyword, returns the height required to display the shortcut's caption

This property/method supports predefined constants and operators/functions as described [here](#).

The following properties customize the format/look the shortcut keys are displayed:

- The [ShortcutKeyFormat](#) property specifies the expression that defines the format (including HTML tags) to display the shortcut keys.

- [Background\(exShortcutKeyAppearance\)](#) / [Background\(exShortcutKeyBackColor\)](#) / [Background\(exShortcutKeyForeColor\)](#) specifies the visual appearance of the shortcut's background / foreground
- The [ShortcutKeyTransparent](#) property Indicates the percent of transparency the shortcut keys are being displayed with.
- The [ShortcutKeyPadding](#) property specifies the UI shortcut key's content padding.
- The [ShortcutKeyAlignH](#) / [ShortcutKeyAlignV](#) defines the alignment of the UI shortcut keys relative to the item that displays it, when the holder arranges the items horizontally.

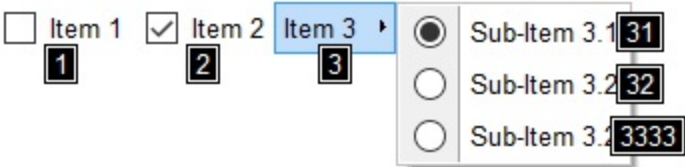
The [ShortcutKeyPressedModifiers](#) property indicates the combination of modifier keys whose shortcut keys are currently visible.

property Ribbon.ShortcutKeyExtPaddingV as String

Specifies the padding to be applied to the item (arranged vertically) that displays the UI shortcut, so the ShortcutKeyAlign property aligns the UI shortcut relative to this.

Type	Description
String	A String expression that specifies the padding to be applied to the item (arranged horizontally) that displays the UI shortcut, so the ShortcutKeyAlign property aligns the UI shortcut relative to this. The list is separated by comma character and indicates left, top, right and bottom exterior padding.

By default, the ShortcutKeyExtPaddingV property is "width/4, height/4 , width/4 ,height/4". The [ShortcutKeyExtPaddingH](#) / [ShortcutKeyExtPaddingV](#) specifies the padding to be applied to the item (arranged horizontally/vertically) that displays the UI shortcut, so the [ShortcutKeyAlign](#) property aligns the UI shortcut relative to this. The [ShortcutKeyVisible](#) property indicates the way the control shows the available keys when user presses any modifier keys such as SHIFT, CTRL or ALT. The [Shortcut](#) property specifies the key combination that the user can press to select the item quickly. The [ShortcutKeysInfo](#) property returns the list of shortcut keys that are currently available.



For instance:

- "4,4,4,4" indicates a fixed exterior padding.
- "width/4, height/4 , width/4 ,height/4" indicates a relative exterior padding.

The [ShortcutKeyExtPaddingV](#) property supports the following predefined keywords:

- **width** keyword, returns the width required to display the shortcut's caption
- **height** keyword, returns the height required to display the shortcut's caption

This property/method supports predefined constants and operators/functions as described [here](#).

The following properties customize the format/look the shortcut keys are displayed:

- The [ShortcutKeyFormat](#) property specifies the expression that defines the format (including HTML tags) to display the shortcut keys.

- [Background\(exShortcutKeyAppearance\)](#) / [Background\(exShortcutKeyBackColor\)](#) / [Background\(exShortcutKeyForeColor\)](#) specifies the visual appearance of the shortcut's background / foreground
- The [ShortcutKeyTransparent](#) property Indicates the percent of transparency the shortcut keys are being displayed with.
- The [ShortcutKeyPadding](#) property specifies the UI shortcut key's content padding.
- The [ShortcutKeyAlignH](#) / [ShortcutKeyAlignV](#) defines the alignment of the UI shortcut keys relative to the item that displays it, when the holder arranges the items horizontally.

The [ShortcutKeyPressedModifiers](#) property indicates the combination of modifier keys whose shortcut keys are currently visible.

property Ribbon.ShortcutKeyFormat as String

Specifies the expression that determines the format (including HTML tags) to display the shortcut keys.

Type	Description
String	A String expression that defines the format (including HTML tags) to display the shortcut keys.

By default, The ShortcutKeyFormat property is "". The ShortcutKeyFormat property specifies the expression that defines the format (including HTML tags) to display the shortcut keys. The [ShortcutKeyVisible](#) property indicates the way the control shows the available keys when user presses any modifier keys such as SHIFT, CTRL or ALT. The [Shortcut](#) property specifies the key combination that the user can press to select the item quickly. The [ShortcutKeysInfo](#) property returns the list of shortcut keys that are currently available.

For instance:

- ""` + caption", displays the shortcut keys with a different font's size.
- ""` + caption", displays the shortcut keys with a different font type
- ""<c><fgcolor 808080>` + sca + `</fgcolor>
<c>Key:` + keys" displays the full modifiers and keys the user has to press to activate / select the item.

The ShortcutKeyFormat property supports the following predefined keywords:

- **caption** keyword defines the keys that the user should press to activate / select the item. The caption keyword includes no modifier keys already pressed. The [ShortcutKeyPressedModifiers](#) property indicates the combination of modifier keys whose shortcut keys are currently visible. For instance "3", specifies that the user can press the 3-key to activate/select the item.
- **sca** keyword defines the modifier keys associated with the shortcut. For instance, "ALT+SHIFT" specifies that the user should press the ALT plus SHIFT combination to select/activate the item
- **keys** keyword defines the keys associated with the shortcut. For instance "3", specifies that the user can press the 3-key to activate/select the item.

This property/method supports predefined constants and operators/functions as described [here](#).

The following properties customize the format/look the shortcut keys are displayed:

- [Background\(exShortcutKeyAppearance\)](#) / [Background\(exShortcutKeyBackColor\)](#) / [Background\(exShortcutKeyForeColor\)](#) specifies the visual appearance of the

shortcut's background / foreground

- The [ShortcutKeyTransparent](#) property Indicates the percent of transparency the shortcut keys are being displayed with.
- The [ShortcutKeyPadding](#) property specifies the UI shortcut key's content padding.
- The [ShortcutKeyExtPaddingH](#) / [ShortcutKeyExtPaddingV](#) specifies the padding to be applied to the item (arranged horizontally/vertically) that displays the UI shortcut, so the [ShortcutKeyAlign](#) property aligns the UI shortcut relative to this.
- The [ShortcutKeyAlignH](#) / [ShortcutKeyAlignV](#) defines the alignment of the UI shortcut keys relative to the item that displays it, when the holder arranges the items horizontally.

The [ShortcutKeyPressedModifiers](#) property indicates the combination of modifier keys whose shortcut keys are currently visible.

property Ribbon.ShortcutKeyPadding as String

Specifies the UI shortcut key's content padding.

Type	Description
String	A String expression that indicates the UI shortcut key's content padding.

By default, the `ShortcutKeyPadding` property is "3,0,3,0". The `ShortcutKeyPadding` property specifies the UI shortcut key's content padding. The [ShortcutKeyVisible](#) property indicates the way the control shows the available keys when user presses any modifier keys such as SHIFT, CTRL or ALT. The [Shortcut](#) property specifies the key combination that the user can press to select the item quickly. The [ShortcutKeysInfo](#) property returns the list of shortcut keys that are currently available.

The following properties customize the format/look the shortcut keys are displayed:

- The [ShortcutKeyFormat](#) property specifies the expression that defines the format (including HTML tags) to display the shortcut keys.
- [Background\(exShortcutKeyAppearance\)](#) / [Background\(exShortcutKeyBackColor\)](#) / [Background\(exShortcutKeyForeColor\)](#) specifies the visual appearance of the shortcut's background / foreground
- The [ShortcutKeyTransparent](#) property Indicates the percent of transparency the shortcut keys are being displayed with.
- The [ShortcutKeyExtPaddingH](#) / [ShortcutKeyExtPaddingV](#) specifies the padding to be applied to the item (arranged horizontally/vertically) that displays the UI shortcut, so the `ShortcutKeyAlign` property aligns the UI shortcut relative to this.
- The [ShortcutKeyAlignH](#) / [ShortcutKeyAlignV](#) defines the alignment of the UI shortcut keys relative to the item that displays it, when the holder arranges the items horizontally.

The [ShortcutKeyPressedModifiers](#) property indicates the combination of modifier keys whose shortcut keys are currently visible.

property Ribbon.ShortcutKeyPressedModifiers as ModifierKeyEnum

Indicates the combination of modifier keys whose shortcut keys are currently visible.

Type	Description
ModifierKeyEnum	A ModifierKeyEnum expression that indicates the combination of modifier keys whose shortcut keys are currently visible.

The ShortcutKeyPressedModifiers property indicates the combination of modifier keys whose shortcut keys are currently visible. The [ShortcutKeyVisible](#) property indicates the way the control shows the available keys when user presses any modifier keys such as SHIFT, CTRL or ALT. The [Shortcut](#) property specifies the key combination that the user can press to select the item quickly. The [ShortcutKeysInfo](#) property returns the list of shortcut keys that are currently available.

property Ribbon.ShortcutKeysInfo as String

Returns the list of shortcut keys that are currently available.

Type	Description
String	A String expression that indicates the list of shortcut keys that are currently available.

The ShortcutKeysInfo property returns the list of shortcut keys that are currently available. You can use the ShortcutKeysInfo property to get the list of shortcut keys that currently the user can press at this moment. The [Shortcut](#) property specifies the key combination that the user can press to select the item quickly. The [ShortcutKeyVisible](#) property indicates the way the control shows the available keys when user presses any modifier keys such as SHIFT, CTRL or ALT.

The ShortcutKeysInfo property may return information as follows:

ALT+1 Item 1[id=20]
ALT+2 Item 2[id=30]
ALT+3 Item 3[id=40]
ALT+3+1 Sub-Item 3.1[id=50]
ALT+3+2 Sub-Item 3.2[id=60]
ALT+3+3+3+3 Sub-Item 3.2[id=70]

property Ribbon.ShortcutKeyTransparent as Long

Indicates the percent of transparency the shortcut keys are being displayed with.

Type	Description
Long	A long expression indicates the percent of transparency the shortcut keys are being displayed with.

By default, the `ShortcutKeyTransparent` property is 0. The `ShortcutKeyTransparent` property Indicates the percent of transparency the shortcut keys are being displayed with. The [ShortcutKeyVisible](#) property indicates the way the control shows the available keys when user presses any modifier keys such as SHIFT, CTRL or ALT. The [Shortcut](#) property specifies the key combination that the user can press to select the item quickly. The [ShortcutKeysInfo](#) property returns the list of shortcut keys that are currently available.

The following properties customize the format/look the shortcut keys are displayed:

- The [ShortcutKeyFormat](#) property specifies the expression that defines the format (including HTML tags) to display the shortcut keys.
- [Background\(exShortcutKeyAppearance\)](#) / [Background\(exShortcutKeyBackColor\)](#) / [Background\(exShortcutKeyForeColor\)](#) specifies the visual appearance of the shortcut's background / foreground
- The [ShortcutKeyPadding](#) property specifies the UI shortcut key's content padding.
- The [ShortcutKeyExtPaddingH](#) / [ShortcutKeyExtPaddingV](#) specifies the padding to be applied to the item (arranged horizontally/vertically) that displays the UI shortcut, so the `ShortcutKeyAlign` property aligns the UI shortcut relative to this.
- The [ShortcutKeyAlignH](#) / [ShortcutKeyAlignV](#) defines the alignment of the UI shortcut keys relative to the item that displays it, when the holder arranges the items horizontally.

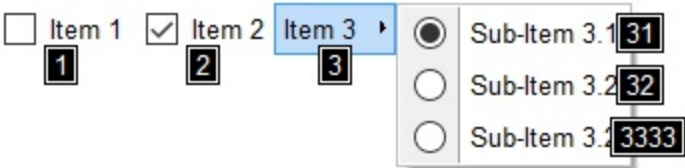
The [ShortcutKeyPressedModifiers](#) property indicates the combination of modifier keys whose shortcut keys are currently visible.

property Ribbon.ShortcutKeyVisible as ShortcutKeyVisibleEnum

Gets or sets a value that specifies whether the control's shortcut keys are visible or hidden.

Type	Description
ShortcutKeyVisibleEnum	A ShortcutKeyVisibleEnum expression that specifies whether the control's shortcut keys are visible or hidden.

By default, the ShortcutKeyVisible property is exShowShortcutKeysToggleDelayed. The ShortcutKeyVisible property indicates the way the control shows the available keys when user presses any modifier keys such as SHIFT, CTRL or ALT. The [Shortcut](#) property specifies the key combination that the user can press to select the item quickly. The [ShortcutKeysInfo](#) property returns the list of shortcut keys that are currently available.



- The following properties customize the format/look the shortcut keys are displayed:
- The [ShortcutKeyFormat](#) property specifies the expression that defines the format (including HTML tags) to display the shortcut keys.
 - [Background\(exShortcutKeyAppearance\)](#) / [Background\(exShortcutKeyBackColor\)](#) / [Background\(exShortcutKeyForeColor\)](#) specifies the visual appearance of the shortcut's background / foreground
 - The [ShortcutKeyTransparent](#) property Indicates the percent of transparency the shortcut keys are being displayed with.
 - The [ShortcutKeyPadding](#) property specifies the UI shortcut key's content padding.
 - The [ShortcutKeyExtPaddingH](#) / [ShortcutKeyExtPaddingV](#) specifies the padding to be applied to the item (arranged horizontally/vertically) that displays the UI shortcut, so the [ShortcutKeyAlign](#) property aligns the UI shortcut relative to this.
 - The [ShortcutKeyAlignH](#) / [ShortcutKeyAlignV](#) defines the alignment of the UI shortcut keys relative to the item that displays it, when the holder arranges the items horizontally.

The [ShortcutKeyPressedModifiers](#) property indicates the combination of modifier keys whose shortcut keys are currently visible.

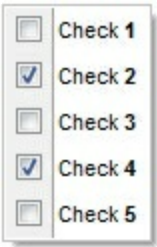
property Ribbon.ShowCheckedAsSelected as ShowCheckedAsSelectedEnum

Specifies whether the checked items shows as selected.

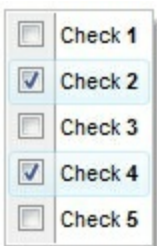
Type	Description
ShowCheckedAsSelectedEnum	A ShowCheckedAsSelectedEnum expression that specifies whether the checked items show as selected.

By default, the ShowCheckedAsSelected property is exDisplayItemCheckDefault. Use the ShowCheckedAsSelected property on non zero, to show the checked items as selected. A checked item is an item with the [Check](#) or [Radio](#) property set on True and the [Checked](#) property is True. The [SelBackColor](#) property indicates the color to show background of the selected / highlighted item. The [AllowToggleRadio](#) property on True, allows a radio button to set on zero (unchecked), if the user clicks twice the radio button. The [ShowCheckedAsSelected](#) property of the Item object specifies whether the individual checked item is shown as selected.

The following screen shot shows the control when the ShowCheckedAsSelected property is exDisplayItemCheckDefault(by default):



The following screen shot shows the control when the ShowCheckedAsSelected property is exDisplayItemCheckHighlight:



property Ribbon.ShowImageList as Boolean

Specifies whether the control's image list window is visible or hidden.

Type	Description
Boolean	A boolean expression that specifies whether the control's image list window is visible or hidden.

By default, the ShowImageList property is True. Use the ShowImageList property to hide the control's images list window. The control's images list window is visible only at design time. Use the [Images](#) method to associate an images list control to the control. The tag can be used in the [Caption](#) property of the [Item](#) object. Also, the [Image](#) property assign an icon to the specified item. Use the [Replacelcon](#) method to add, remove or clear icons in the control's images collection.

property Ribbon.ShowPopupArrow(ItemHighlited as Boolean) as ShowPopupArrowEnum

Indicates the type of the arrow to be shown when the item displays the sub-menu.

Type	Description
ItemHighlited as Boolean	A Boolean expression that specifies whether the arrow is shown to the selected/highlighted item. <i>In VFP, you should use 0 or 1, instead .F. or .T.</i>
ShowPopupArrowEnum	A ShowPopupArrowEnum expression that specifies the arrow to be shown on an item that displays a submenu.

By default, the ShowPopupArrow(True) property is exShowPopupArrowLight, and the ShowPopupArrow(False) property is exShowPopupArrowDark. In other words, when the item is selected/highlighted a light arrow is displayed, while when the item it is not selected/highlighted a dark arrow is displayed. Use the ShowPopupArrow property to specify how the item with sub-menu displays the popup arrow.



property Ribbon.ShowPopupEffect as ShowPopupEffectEnum

Specifies the effect to show the popup menu when clicking an item, such as scrolling, lighting up, and so on.

Type	Description
ShowPopupEffectEnum	A ShowPopupEffectEnum expression that specifies effect to be applied when the user opens the menu or a sub-menu.

By default, ShowPopupEffect property is exShowPopupLightUp. Use the ShowPopupEffect property to disable the effect to be applied when the user opens the menu or any sub-menu.

method Ribbon.ShowToolTip (ToolTip as String, [Title as Variant], [Alignment as Variant], [X as Variant], [Y as Variant])

Shows the specified tooltip at given position.

Type	Description
ToolTip as String	<p>The ToolTip parameter can be any of the following:</p> <ul style="list-style-type: none">• 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	<p>The Title parameter can be any of the following:</p> <ul style="list-style-type: none">• 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	<p>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.</p> <p>The Alignment parameter can be one of the following:</p> <ul style="list-style-type: none">• 0 - exTopLeft• 1 - exTopRight• 2 - exBottomLeft• 3 - exBottomRight• 0x10 - exCenter• 0x11 - exCenterLeft• 0x12 - exCenterRight• 0x13 - exCenterTop• 0x14 - exCenterBottom <p>By default, the tooltip is aligned relative to the top-left corner (0 - exTopLeft).</p>

Specifies the horizontal position to display the tooltip as one of the following:

- missing (VT_EMPTY, VT_ERROR type), indicates that the tooltip is shown on its default position / current cursor position (ignored)
- -1, indicates the current horizontal position of the cursor (current x-position)
- a numeric expression that indicates the horizontal screen position to show the tooltip (fixed screen x-position)
- a string expression that indicates the horizontal displacement relative to default position to show the tooltip (moved)

X as Variant

Specifies the vertical position to display the tooltip as one of the following:

- missing (VT_EMPTY, VT_ERROR type), indicates that the tooltip is shown on its default position / current cursor position (ignored)
- -1, indicates the current vertical position of the cursor (current y-position)
- a numeric expression that indicates the vertical screen position to show the tooltip (fixed screen y-position)
- a string expression that indicates the vertical displacement relative to default position to show the tooltip (displacement)

Y as Variant

Use the ShowToolTip method to display a custom tooltip at specified position or to update the object's tooltip, title or position. You can call the ShowToolTip method during the [MouseMove](#) event. Use the [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. The [ToolTipDelay](#) property specifies the time in ms that passes before the ToolTip appears. Use the [ToolTipWidth](#) property to specify the width of the tooltip window. Use the [ToolTipFont](#) property to change the tooltip's font. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color.

For instance:

- [ShowToolTip\(<null>, <null>, , +8, +8\)](#), shows the tooltip of the object moved relative

to its default position

- `ShowToolTip(<null>`,`new title`)`, adds, changes or replaces the title of the object's tooltip
- `ShowToolTip(`new content`)`, adds, changes or replaces the object's tooltip
- `ShowToolTip(`new content`,`new title`)`, shows the tooltip and title at current position
- `ShowToolTip(`new content`,`new title`,`+8`,`+8`)`, shows the tooltip and title moved relative to the current position
- `ShowToolTip(`new content`,``,`128,128`)`, displays the tooltip at a fixed position
- `ShowToolTip(``,``)`, hides the tooltip

The ToolTip parameter supports the built-in HTML format like follows:

- ` ... ` displays the text in **bold**
- `<i> ... </i>` displays the text in *italics*
- `<u> ... </u>` underlines the text
- `<s> ... </s>` Strike-through text
- `<a id;options> ... ` displays an [anchor](#) element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link. The `<a>` element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.
- ` ... ` 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
- **number[:width]** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **key[:width]** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&**; (&), **<**; (<), **>**; (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the **€** displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display **bold** in HTML caption you can use **bold**;
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated **</off>** tag is found. You can use the **<off offset>** HTML tag in combination with the **** to define a smaller or a larger font to be displayed. For instance: "Text with **<off 6>subscript**" displays the text such as: Text with subscript The "Text with **<off -6>superscript**" displays the text such as: Text with superscript
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or **<fgcolor>** defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The **** HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The **<gra>** with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "**<gra FFFFFFFF;1;1>gradient-center</gra>**" generates the following picture:

gradient-center

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

height of the font. For instance the "<out 000000>

<fgcolor=FFFFFF>outlined</fgcolor></out>" generates the following picture:

outlined

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

shadow

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

outline anti-aliasing

property Ribbon.Template as String

Specifies the control's template.

Type	Description
String	A string expression that indicates the control's template.

The control's template uses the X-Script language to initialize the control's content. Use the Template property page of the control to update the control's Template property. Use the Template property to execute code by passing instructions as a string (template string). Use the [ExecuteTemplate](#) property to execute a template script and gets the result. The [ToTemplate](#) property generates x-script from the control's content.

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 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 Ribbon.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 Ribbon.TemplatePut (NewVal as Variant)

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
NewVal as Variant	A string expression that indicates the Dim declaration, or any Object expression to be assigned to previously declared variables.

The TemplatePut method / [TemplateDef](#) property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters. It is known that programming languages such as **dBASE Plus** or **XBasic from AlphaFive**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters) = Value*, so our controls provide an alternative using the TemplateDef / TemplatePut method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be [Template](#) or [ExecuteTemplate](#) property which can use the variable a and b being defined previously.

The [TemplateDef](#), TemplatePut, [Template](#) and [ExecuteTemplate](#) support x-script language (Template script of the Exontrols), like explained bellow:

The Template or x-script is composed by lines of instructions. Instructions are separated by "\n\r" (newline characters) or ";" character. The ; character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas.* (Sample: Dim h, h1, h2)
- variable = property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas.* (Sample: h = InsertItem(0,"New Child"))
- property(list of arguments) = value *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- method(list of arguments) *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- { *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- } *Ending the object's context*
- object. property(list of arguments).property(list of arguments).... *The .(dot) character splits the object from its property. For instance, the*

Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.

The x-script may use constant expressions as follows:

- *boolean* expression with possible values as *True* or *False*
- *numeric* expression may start with 0x which indicates a hexa decimal representation, else it should start with a digit, or +/- followed by a digit, and . is the decimal separator. *Sample: 13 indicates the integer 13, or 12.45 indicates the double expression 12,45*
- *date* expression is delimited by # character in the format #mm/dd/yyyy hh:mm:ss#. *Sample: #31/12/1971# indicates the December 31, 1971*
- *string* expression is delimited by " or ` characters. If using the ` character, please make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also, the template or x-script code may support general functions as follows:

- **Me** property indicates the original object.
- **RGB(R,G,B)** property retrieves an RGB value, where the R, G, B are byte values that indicate the R G B values for the color being specified. For instance, the following code changes the control's background color to red: *BackColor = RGB(255,0,0)*
- **LoadPicture(file)** property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.
- **CreateObject(progID)** property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.

property Ribbon.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 [ToolTip](#) property to assign a tooltip to an item. Use the [ToolTipFont](#) property or HTML element to assign a new font for tooltips.

property Ribbon.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. You can use the ** HTML element, in the tooltip's description to assign a different font for portions of text. Use the [ToolTip](#) property to assign a tooltip to an item

property Ribbon.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 [ToolTip](#) property to assign a tooltip to an item

property Ribbon.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 [ToolTip](#) property to assign a tooltip to an item

property Ribbon.ToString as String

Loads or saves the Items collection using string representation (shortcut of Items.ToString property).

Type	Description
String	A String expression that specifies the items to be added. The list of items is separated by , (comma) character, while sub-menus are include between () parenthesis. The [] brackets indicates the options to be applied on the item

The ToString property of the control is equivalent with the [ToString](#) property of the [Items](#) object.

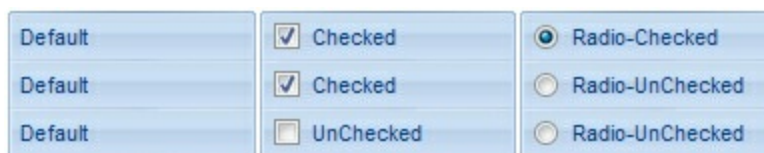
The ToString syntax in BNF notation:

```
<ToString> ::= <ITEMS>
<ITEMS> ::= <ITEM>["("<ITEMS>")"][","<ITEMS>]
<ITEM> ::= <CAPTION>[<OPTIONS>]
<OPTIONS> ::= "["<OPTION>"] "["<OPTIONS>"]
<OPTION> ::= <PROPERTY>["="<VALUE>]
<PROPERTY> ::= "img" | "himg" | "sep" | "id" | "typ" | "group" | "chk" | "button" | "align" |
"spchk" | "show" | "rad" | "dis" | "showdis" | "bld" | "itl" | "stk" | "und" | "bg" | "fg" | "editttype"
| "edit" | "mask" | "border" | "editwidth" | "captionwidth" | "height" | "grp" | "tfi" | "ttp" | "min" | |
|max" | "tick" | "freq" | "ticklabel" | "small" | "large" | "spin" | "ettp" | "float" | "close" | "local" |
| "popupapp" | "itemspad" | "itemsbg" | "itemsbghot" | "itemsbgext" | "visible" | "tab" | "pad" |
| "bghot" | "bgssel" | "bgsselhot" | "arrow" | "popupalign" | "popupoffset" | "popupat"
```

where the <CAPTION> is the HTML caption to be shown on the context menu item. The <VALUE> indicates the value of giving property.

- id=<VALUE>, where <VALUE> is an integer expression, that indicates the identifier of the item.
- bg=<VALUE>, specifies the item's background color, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or an integer expression to that refers an EBN object.
- bghot=<VALUE>, specifies the item's background color, while the cursor hovers the item, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or an integer expression to that refers an EBN object.
- bgssel=<VALUE>, specifies the item's background color, while the item is checked/selected, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or an integer expression to that refers an EBN object.

- `bgselfhot=<VALUE>`, specifies the item's background color, while the item is checked/selected and the cursor hovers it, where `<VALUE>` could be a RGB expression (`RGB(RR,GG,BB)`, where `RR` is the red value, the `GG` is the green value, and the `BB` is the blue value), or an integer expression to that refers an EBN object.
- `fg=<VALUE>`, specifies the item's foreground color, where `<VALUE>` could be a RGB expression (`RGB(RR,GG,BB)`, where `RR` is the red value, the `GG` is the green value, and the `BB` is the blue value), or a integer expression.
- `sep`, specifies an separator item
- `dis`, specifies a disabled item
- `showdis=<VALUE>`, where `<VALUE>` could be **0** for regular or **not zero** to specify whether the item shows as disabled, but it is still enabled
- `bld`, specifies that the item appears in bold
- `itl`, specifies that the item appears in italics
- `stk`, specifies that the item appears as strikeouts
- `und`, specifies that the item is underlined
- `align=<VALUE>`, where `<VALUE>` could be one of the following:
 - **0** (left), to align the item's caption to the left
 - **1** (center), to center the item's caption
 - **2** (right), to align the item's caption to the right
- `captionwidth=<VALUE>`, specifies the width to show the HTML caption of the item. where `<VALUE>` could be a integer expression. A negative value indicates that no limitation is applied to the item's caption, so no truncate caption is shown
- `height=<VALUE>`, specifies the height to show the item, where `<VALUE>` could be a positive integer expression
- `pad=<VALUE>`, specifies the padding (space between the menu border and the item content) to display the item. The `<VALUE>` is a list of coordinates such as `left,top,right,bottom`
- `img=<VALUE>`, where `<VALUE>` is an integer expression, that indicates the index of the icon being displayed for the item.
- `himg=<VALUE>`, where `<VALUE>` indicates the key of the picture to be displayed for the item.

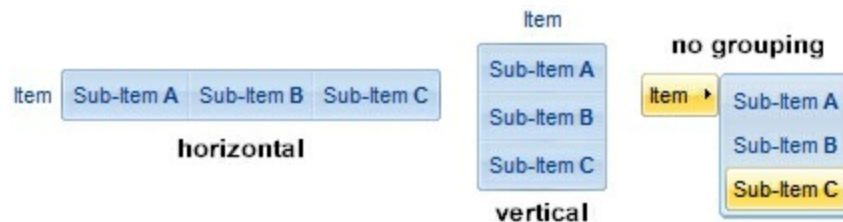


- `typ=<VALUE>`, where `<VALUE>` could be one of the following:
 - **0** for default/regular items (no check/radio button is associated with the item),
 - **1** for items that display a check/box (chk),
 - **2** to display radio buttons (rad)
- `chk[=<VALUE>]`, where `<VALUE>` could be **0** for unchecked, or **not zero** for checked. The `chk` option makes the item to display a check box. If the `<VALUE>` is missing the item still displays an un-checked check box.

- `rad=<VALUE>`, where `<VALUE>` could be **0** for unchecked radio button or **not zero** to for checked radio button. Use the `grp` option to define the group of radio where this button should be associated, If no group of radio buttons is required, the `grp` could be ignored.
- `grp=<VALUE>`, defines the radio group. It should be used when you define more groups of radio buttons. A group of radio buttons means that only one item could be checked at one time. The `rad` option specifies that the item displays a radio button. Use the `grp` option to define the group of radio where this button should be associated, If no group of radio buttons is required, the `grp` could be ignored. The `<VALUE>` could be any integer expression.



- `show=<VALUE>`, where `<VALUE>` could be **0** for regular or **not zero** to specify whether the checked item shows as selected
- `spchk=<VALUE>`, where `<VALUE>` could be **0** for regular or **not zero** to specify whether the item's sub menu is shown only if the item is checked.



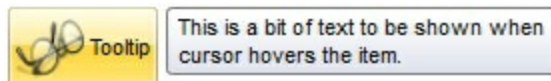
- `group=<VALUE>`, where `<VALUE>` could be a bit-or combination (+) of the following values:
 - **0** (`exNoGroupPopup`), No grouping is performed on the sub-menu, so the sub-items are shown to a float popup,
 - **1** (`exGroupPopup`), Groups and displays the sub-menu items on the current item, arranged from left to right/horizontally
 - **2** (`exNoGroupPopupFrame`), Prevents showing the frame around each grouping item.
 - **4** (`exGroupPopupCenter`), Shows the grouping popup aligned to the center of the current item.
 - **8** (`exGroupPopupRight`), Shows the grouping popup aligned to the right of the current item.
 - **16** (`exGroupPopupEqualWidth`), Shows the items that make the group of the same width
 - **32** (`exGroupPopupEqualHeight`), Shows the items that make the group of the same height
 - **64** (`exGroupPopupFrameSolidBox`), Shows a solid frame around the grouped

items

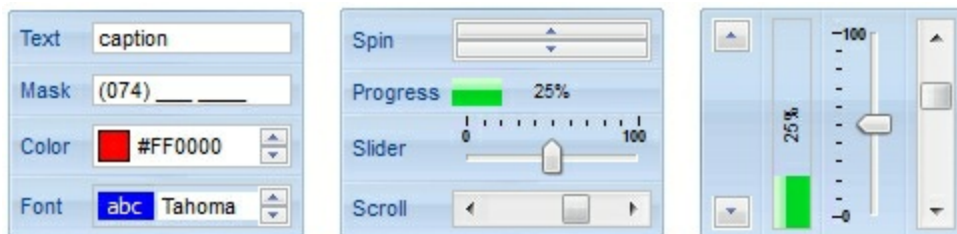
- **128** (exGroupPopupFrameThickBox), Shows a solid thick-frame around the grouped items
- **256** (exGroupPopupVertical), Groups and displays the sub-menu items on the current item, arranged from top to bottom/vertically



- button=<VALUE>, where <VALUE> could be a bit-or combination (+) of the following values.
 - **0** (exShowAsButtonNone), No button is shown,
 - **1** (exShowAsButton), Shows the item as a button
 - **2** (exShowAsButtonAutoSize), Fits the button to cover the item's caption instead showing on the entire item
 - **17** (exShowAsSelectButton), Shows the item as a select button, which is composed by two-fields, one indicates the default button, while the second field specifies the drop down button that displays the items in the current's sub-menu collection. The drop down button is shown to the right-side of the default button. The item must have a submenu, else no drop down is displayed.
 - **273** (exShowAsSelectButtonBottom), Shows the item as a select button, which is composed by two-fields, one indicates the default button, while the second field specifies the drop down button that displays the items in the current's sub-menu collection. The drop down button is shown to the bottom-side of the default button. The item must have a submenu, else no drop down is displayed.



- ttp=<VALUE>, defines the item's tooltip. The <VALUE> could be any HTML string expression. The item's tooltip is shown when the user hovers the item.



- edittype=<VALUE>, associates an edit field to the item, where <VALUE> could be a combination of one or more of the following values:
 - **0** (exItemDisableEdit), No editor is assigned to the current item.
 - **1** (exItemEditText), A text-box editor is assigned to the current item.
 - **2** (exItemEditMask), A masked text-box editor is assigned to the current item.

- **3** (`exltemEditSlider`), A slider editor is assigned to the current item. This can be combined with 1024.
- **4** (`exltemEditProgress`), A progress editor is assigned to the current item. This can be combined with 1024.
- **5** (`exltemEditScrollBar`), A scrollbar editor is assigned to the current item. This can be combined with 1024.
- **6** (`exltemEditColor`), A color editor is assigned to the current item.
- **7** (`exltemEditFont`), A font editor is assigned to the current item.
- **256** (`exltemEditReadOnly`), specifies that the item's editor is shown as disabled. This value could be combined with one of the values from 0 to 7 or 512
- **512** (`exltemEditSpin`), A spin editor is assigned to the current item. This value could be combined with one of the values from 0 to 7 or 256
- **1024** (`exltemEditVertical`), The editor is shown vertically rather than horizontally. This value has effect for `exltemEditSlider`, `exltemEditProgress` or `exltemEditScrollBar`
- `edit=<VALUE>`, specifies the caption to be shown in the item's edit field, where `<VALUE>` could be any string
- `mask=<VALUE>`, specifies the mask to be applied on a masked editor. This option is valid for `exltemEditMask` edit. Use the float option to allow masking floating point numbers. See [Masking](#) for more information about `<VALUE>` of the mask option. See [Masking Float](#) for more information about `<VALUE>` if the float option is used.
- `float=<VALUE>`, Specifies whether the mask field masks a floating point number. This option is valid for `exltemEditMask` edit. See [Masking Float](#) for more information about `<VALUE>` of mask option, if the float option is used. The `<VALUE>` could be **0** for standard masking field or **not zero** to specify that the field is masking a floating point.
- `border=<VALUE>`, specifies the border to be shown on the item's edit field, where `<VALUE>` could be one of the following:
 - **0** (`exEditBorderNone`), No border is shown.
 - **-1** (`exEditBorderInset`), shows an inset border
 - **1** (`exEditBorderSingle`), shows a frame border
- `editwidth=<VALUE>`, specifies the width to show the edit field inside the item, where `<VALUE>` could be a integer expression. A negative value indicates that the field goes to the end of the item
- `min=<VALUE>`, defines the minimum value of the edit field. The `<VALUE>` could be any integer expression, and specifies the minimum value for any slider, progress, scroll, spin, or range editor.
- `max=<VALUE>`, defines the maximum value of the edit field. The `<VALUE>` could be any integer expression, and specifies the maximum value for any slider, progress, scroll, spin, or range editor.
- `tick=<VALUE>`, defines where the ticks of the slider edit appear. This option is valid for `exltemEditSlider` edit. The `<VALUE>` could be one of the following values:
 - **0** (`exBottomRight`), The ticks are displayed on the bottom/right side.
 - **1** (`exTopLeft`), The ticks are displayed on the top/left side.

- **2** (`exBoth`), The ticks are displayed on the both side.
- **3** (`exNoTicks`), No ticks are displayed.
- `freq=<VALUE>`, indicates the ratio of ticks on the slider edit. This option is valid for `exItemEditSlider` edit. The `<VALUE>` could be a positive integer expression.
- `ticklabel=<VALUE>`, indicates the HTML label to be displayed on slider's ticks. This option is valid for `exItemEditSlider` edit. See [Tick Label Expression](#) for more information about `<VALUE>` of the `ticklabel` option.
- `small=<VALUE>`, indicates the amount by which the edit's position changes when the user presses the arrow key (left, right, or button). This option is valid for `exItemEditSlider`, `exItemEditScrollBar` edit. The `<VALUE>` could be a positive integer expression.
- `large=<VALUE>`, indicates the amount by which the edit's position changes when the user presses the CTRL + arrow key (CTRL + left, CTRL + right). This option is valid for `exItemEditSlider`, `exItemEditScrollBar` edit. The `<VALUE>` could be a positive integer expression.
- `spin=<VALUE>`, specifies the step to advance when user clicks the editor's spin.. This option is valid for `exItemEditSpin` edit. The `<VALUE>` could be a positive integer expression.
- `ettp=<VALUE>`, specifies the HTML tooltip to be shown when the item's value is changed. This option is valid for `exItemEditSlider/exItemEditScrollBar` edit. The `<VALUE>` could be any string expression, including built-in HTML tags
- `arrow=<VALUE>`. The `<VALUE>` could be **0** for hiding the arrow or **not zero** to show the arrow. Indicates whether an item that has a sub-menu shows or hides its popup arrow. If the `<VALUE>` is missing, the item shows no arrow.
- `local=<VALUE>`. The `<VALUE>` could be **0** for standard popup or **not zero** to specify that the field is a local popup. Specifies whether the item's popup is shown as local. Clicking any item inside a local popup makes the popup itself to close including all its descendent sub-menus, without closing any ascendant sub-menus.
- `close=<VALUE>`, Specifies the way the hosting menu is closed when the user clicks the item. If the close flag is missing, the `<VALUE>` is 3 (`exCloseOnNonClickable`), by default. The `<VALUE>` could be one of the following values:
 - **0** (`exCloseOnClick`), The popup menu is closing when the user clicks the item.
 - **1** (`exCloseOnDbClick`), The popup menu is closing when the user double clicks the item.
 - **2** (`exCloseOnClickOutside`), The popup menu is closing when the user clicks outside of the menu.
 - **3** (`exCloseOnNonClickable`), The popup menu is closing when the user clicks a non-clickable item (regular items). The non-clickable items is any item that's not a separator, popup, disabled or check or radio items, clicking a check-box item will makes the check box to change its state instead closing the context menu.
- `popupapp=<VALUE>` indicates the visual appearance of the item's submenu when the popup is shown. The `<VALUE>` could be a predefine value like shown bellow, or an

integer expression that refers an EBN object.

- **0** (NoBorder)
- **1** (FlatBorder)
- **2** (SunkenBorder)
- **3** (RaisedBorder)
- **4** (EtchedBorder)
- **5** (BumpBorder)
- **6** (ShadowBorder)
- **7** (InsetBorder)
- **8** (SingleBorder)
- itemsbg=<VALUE>, specifies the items background color, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or an integer expression to that refers an EBN object.
- itemsbgshot=<VALUE>, specifies the items background color, while the cursor hovers the items, where <VALUE> could be a RGB expression (RGB(RR,GG,BB), where RR is the red value, the GG is the green value, and the BB is the blue value), or an integer expression to that refers an EBN object.
- popupalign=<VALUE>, Indicates how the item's sub-menu is aligned relative to the parent item. The popupalign has no effect for an item that displays a select- button. The <VALUE> could be a combination of one or more of the following values:
 - **0** (exShowPopupAlignNone), The popup menu is shown on top of the item, aligned to the left (no down and right, so up and left)
 - **1** (exShowPopupAlignDown), The popup menu is shown down. If missing, the popup menu is shown up.
 - **2** (exShowPopupAlignRight), The popup menu is shown aligned to the right, else if missing, the popup menu is shown aligned to the left.
- popupat=<VALUE>, specifies the identifier of the item where the current item's submenu/popup is displayed. The <VALUE> could be any integer expression. If there is no identifier with giving value, the option has no effect.
- popupoffset=<VALUE>, specifies the offset (horizontal,vertical) to display the item's submenu/popup relative to its default position.
- itemspad=<VALUE>, specifies the padding (space between the menu border and the item content) to display the items. The <VALUE> is a list of coordinates such as left,top,right,bottom
- visible=<VALUE>, specifies the maximum number of visible items at one time, where the <VALUE> could be any integer expression.
- tab=<VALUE>, specifies the identifier of the item/tab where the current group-popup is shown instead. The <VALUE> could be any integer expression. If there is no identifier with giving value, the option has no effect.
- itemsbgext=<VALUE>, indicates additional colors, text, images that can be displayed on the items background using the [EBN String Format](#). The <VALUE> should be in [EBN](#)

[String Format](#). For instance, `[itemsbgext=bottom[2],bottom[16,text=`</fgcolor><fgcolor 6D6AAA>Views</fgcolor><fgcolor A0A0A0>`,align=0x21]]`, shows the Views aligned to the bottom, with a different foreground color.

Masking, (mask option)

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

`!(999) 000 0000;1;;select=1,empty,overtypewarning=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 `overtypemodewarning` 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;overtypewarning=<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
- **ŧ**, (ALT + 175) causes the characters that follow to be converted to uppercase, until **Ŧ**(ALT + 174) is found.
- **Ŧ**, (ALT + 174) causes the characters that follow to be converted to lowercase, until **ŧ**(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)
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 overtyping option, it indicates that the user can toggle the insert/over-type mode using the Insert key. For instance, the "###:###;0;inserttype,overtyping", 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.
- **overtyping**, 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;overtyping,inserttype", indicates that the field enter in over-type mode, and insert-type mode is allowed. The "###:###;0;overtyping", 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.

- **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,overtyp"`, 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,overtyp,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

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.

Masking-Float, (mask, float option)

The [mask=<VALUE>] property may indicate the followings, if the [float=-1] is present

- **negative number**: if the first character in the mask is - (minus) the control supports negative numbers. Pressing the - key will toggle the sign of the number. The + sign is never displayed.
- **decimal symbol**: the last character that's different than # (digit), or 0 (zero) indicates the decimal symbol. If it is not present the control mask a floating point number without decimals.
- **thousand symbol**: the thousand symbol is the last character that's not a # (digit), 0 (zero) or it is not the decimal symbol as explained earlier, if present.
- the maximum **number of decimals** in the number (the # or 0 character after the decimal symbol)
- the maximum number of digits in the integer part (the number of # or 0 character before decimal symbol)
- the **0** character indicates **a leading-zero**. The count of 0 (zero) characters before decimal character indicates the leading-zero for integer part of the control, while the count of 0 (zero) characters after the decimal separator indicates the leading-zero for decimal part of the control. For instance, the Mask on "-###,###,##0.00", while the control's Text property is 1, the control displays 1.00, if 1.1 if displays 1.10, and if empty, the 0.00 is displayed.

If the <VALUE> property is empty, the control takes the settings for the regional options like: Decimal Symbol , No. of digits after decimal, Digit grouping symbol.

Here are few samples:

The <VALUE>"-###.###.##0,00" filter floating point numbers a number for German settings ("," is the decimal sign, "." is the thousands separator). This format displays leading-zeros.

The <VALUE>"-###.###.###,##" filter floating point numbers a number for German settings ("," is the decimal sign, "." is the thousands separator)

The <VALUE>"-###,###,###.##" filter floating point numbers a number for English settings ("." is the decimal sign, "," is the thousands separator)

The <VALUE>"####" indicates a max-4 digit number (positive) without a decimal symbol and without digit grouping

The <VALUE>"-##.##" filters a floating point number from the -99.9 to 99.9 ("." is the decimal sign, no thousands separator)

The <VALUE>"#,###.##" filters a floating point number from the 0 to 9,999.99 with digit grouping ("." is the decimal sign, "," is the thousands separator).

Tick Label Expression, (ticklabel option)



For instance:

- "value", shows the values for each tick.
- "(value=current ? '<fgcolor=FF0000>' : ") + value", shows the current slider's position with a different color and font.
- "value = current ? value : """, shows the value for the current tick only.
- "(value = current ? '' : ") + (value array 'ab bc cd de ef fg gh hi ij jk kl' split ' ')" displays different captions for slider's values.

The The <VALUE> of [ticklabel] option is a formatted expression which result may include the [HTML](#) tags.

The The <VALUE> of [ticklabel] option indicates a formatting expression that may use the following predefined keywords:

- **value** gets the slider's position to be displayed
- **current** gets the current slider's value.
- **vmin** gets the slider's minimum value.
- **vmax** gets the slider's maximum value.
- **smin** gets the slider's selection minimum value.
- **smax** gets the slider's selection maximum value.

The supported binary arithmetic operators are:

- * (multiplicity operator), priority 5
- / (divide operator), priority 5
- **mod** (remainder operator), priority 5
- + (addition operator), priority 4 (concatenates two strings, if one of the operands is of string type)
- - (subtraction operator), priority 4

The supported unary boolean operators are:

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

The supported binary boolean operators are:

- **or** (or operator), priority 2
- **and** (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 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 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 as time consuming as the equivalent *or* version is, so when you have large number of constant elements it is recommended using the *in* operator. Shortly, if the collection of elements has 1000 elements the *in* operator could take up to 8 operations in order to find if an element fits the set, else if the *or* statement is used, it could take up to 1000 operations to check, so by far, the *in* operator could save time on finding elements within a collection.

- ***switch*** (*switch operator*), returns the value being found in the collection, or a predefined value if the element is not found (default). The syntax for *switch* operator is

"expression switch (default,c1,c2,c3,...,cn)"

, where the *c1*, *c2*, ... are constant elements, and the *default* is a constant element being returned when the element is not found in the collection. The constant elements could be numeric, date or string expressions. The equivalent syntax is "%0 = c 1 ? c 1 : (%0 = c 2 ? c 2 : (... ? . : default))". The *switch* operator is very similar with the *in* operator excepts that the first element in the switch is always returned by the statement if the element is not found, while the returned value is the value itself instead -1. For instance, the "%0 switch ('not found',1,4,7,9,11)" gets 1, 4, 7, 9 or 11, or 'not found' for any other value. As the *in* operator the *switch* operator uses binary searches for fitting the element, so it is quicker than the *if* (immediate if operator) alternative.

- ***case()*** (*case operator*) returns and executes one of *n* expressions, depending on the evaluation of the expression (*IIF* - immediate IF operator is a binary *case()* operator). The syntax for *case()* operator is:

"expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)"

If the *default* part is missing, the *case()* operator returns the value of the expression if it is not found in the collection of cases (*c1*, *c2*, ...). For instance, if the value of expression is not any of *c1*, *c2*, the *default_expression* is executed and returned. If the value of the expression is *c1*, then the *case()* operator executes and returns the *expression1*. The *default*, *c1*, *c2*, *c3*, ... must be constant elements as numbers, dates or strings. For instance, the "*date(shortdate(value)) case (default:0 ; #1/1/2002#:1 ; #2/1/2002#:1 ; #4/1/2002#:1 ; #5/1/2002#:1)*" indicates that only #1/1/2002#, #2/1/2002#, #4/1/2002# and #5/1/2002# dates returns 1, since the others returns 0. For instance the following sample specifies the hour being non-working for specified dates: "*date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)*" statement indicates the working hours for dates as follows:

- - #4/1/2009#, from hours 06:00 AM to 12:00 PM
 - #4/5/2009#, from hours 07:00 AM to 10:00 AM and hours 03:00PM, 04:00PM, 06:00PM and 10:00PM
 - #5/1/2009#, from hours 12:00 AM to 08:00 AM

The *in*, *switch* and *case()* use binary search to look for elements so they are faster than using *if* and *or* expressions.

Obviously, the priority of the operations inside the expression is determined by () parenthesis and the priority for each operator.

The supported conversion unary operators are:

- **type** (unary operator) retrieves the type of the object. For instance `type(%0) = 8` specifies the cells that contains string values.

Here's few predefined types:

- 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
- **str** (unary operator) converts the expression to a string
- **dbl** (unary operator) converts the expression to a number
- **date** (unary operator) converts the expression to a date, based on your regional settings
- **dateS** (unary operator) converts the string expression to a date using the format

Other known operators for numbers are:

- **int** (unary operator) retrieves the integer part of the number
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2
- 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

Other known operators for strings are:

- **len** (unary operator) retrieves the number of characters in the string
- **lower** (unary operator) returns a string expression in lowercase letters
- **upper** (unary operator) returns a string expression in uppercase letters
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names
- **ltrim** (unary operator) removes spaces on the left side of a string
- **rtrim** (unary operator) removes spaces on the right side of a string
- **trim** (unary operator) removes spaces on both sides of a string
- **startswith** (binary operator) specifies whether a string starts with specified string
- **endwith** (binary operator) specifies whether a string ends with specified string
- **contains** (binary operator) specifies whether a string contains another specified string
- **left** (binary operator) retrieves the left part of the string
- **right** (binary operator) retrieves the right part of the string
- a **mid** b (binary operator) retrieves the middle part of the string a starting from b (1 means first position, and so on)
- a **count** b (binary operator) retrieves the number of occurrences of the b in a
- a **replace** b **with** c (double binary operator) replaces in a the b with c, and gets the result.
- a **split** b, 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

Other known operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel.
- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance the timeF(1:23 PM) returns "13:23:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel.
- **shortdateF** (unary operator) formats a date as a date string using the "MM/DD/YYYY" format. For instance the shortdateF(December 31, 1971 11:00 AM) returns "12/31/1971".
- **dateF** (unary operator) converts the date expression to a string expression in "MM/DD/YYYY HH:MM:SS" format.

- **longdate** (unary operator) formats a date as a date string using the long date format, as specified in the control's panel.
- **year** (unary operator) retrieves the year of the date (100,...,9999)
- **month** (unary operator) retrieves the month of the date (1, 2,...,12)
- **day** (unary operator) retrieves the day of the date (1, 2,...,31)
- **yearday** (unary operator) retrieves the number of the day in the year, or the days since January 1st (0, 1,...,365)
- **weekday** (unary operator) retrieves the number of days since Sunday (0 - Sunday, 1 - Monday,..., 6 - Saturday)
- **hour** (unary operator) retrieves the hour of the date (0, 1, ..., 23)
- **min** (unary operator) retrieves the minute of the date (0, 1, ..., 59)
- **sec** (unary operator) retrieves the second of the date (0, 1, ..., 59)

The The <VALUE> of [ticklabel] option can display labels using the following built-in HTML tags:

- **** displays the text in **bold**.
- **<i></i>** displays the text in *italics*.
- **<u></u>** underlines the text.
- **<s></s>** Strike-through text
- **** displays portions of text with a different font and/or different size. For instance, the bit draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, bit displays the bit text using the current font, but with a different size.
- **<fgcolor=RRGGBB></fgcolor>** displays text with a specified **foreground** color. The RR, GG or BB should be hexa values and indicates red, green and blue values.
- **<bgcolor=RRGGBB></bgcolor>** displays text with a specified **background** color. The RR, GG or BB should be hexa values and indicates red, green and blue values.
- **
** a forced line-break
- **<solidline>** The next line shows a solid-line on top/bottom side. If has no effect for a single line caption.
- **<dotline>** The next line shows a dot-line on top/bottom side. If has no effect for a single line caption.
- **<upline>** The next line shows a solid/dot-line on top side. If has no effect for a single line caption.
- **<r>** Right aligns the text
- **<c>** Centers the text
- **number[:width]** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method

to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.

- **key[:width]** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- **&** glyph characters as **&**; (&), **<**; (<), **>**; (>), **&qout;** (") and **&#number** (the character with specified code), For instance, the **€** displays the EUR character, in UNICODE configuration. The **&** ampersand is only recognized as markup when it is followed by a known letter or a # character and a digit. For instance if you want to display **bold** in HTML caption you can use **bold**;

EBN String Format, (itemsbgext option)

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

```
<EBN> ::= <elements> | <root> "(" [<elements>] ")"
<elements> ::= <element> [ "," <elements> ]
<root> ::= "root" [ <attributes> ] | [ <attributes> ]
<element> ::= <anchor> [ <attributes> ] [ "(" [<elements>] ")" ]
<anchor> ::= "none" | "left" | "right" | "client" | "top" | "bottom"
<attributes> ::= "[" [<client> "," <attribute> [ "," <attributes> ] "]"
<client> ::= <expression> | <expression> "," <expression> "," <expression> ","
<expression>
<expression> ::= <number> | <number> "%"
<attribute> ::= <bgcolor> | <text> | <wordwrap> | <align> | <pattern> |
<patterncolor> | <frame> | <framethick> | <data> | <others>
<equal> ::= "="
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<decimal> ::= <digit> <decimal>
<hexadigit> ::= <digit> | "A" | "B" "C" | "D" | "E" "F"
<hexa> ::= <hexadigit> <hexa>
<number> ::= <decimal> | "0x" <hexa>
<color> ::= <rgbcolor> | number
<rgbcolor> ::= "RGB" "(" <number> "," <number> "," <number> ")"
<string> ::= "\"" <characters> "\"" | "'" <characters> "'" | "<characters>"
```

```

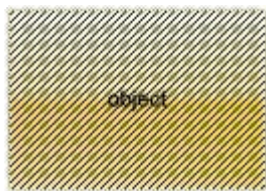
<characters> ::= <char>|<characters>
<char> ::= <any_character_excepts_null>
<backcolor> ::= "back" <equal> <color>
<text> ::= "text" <equal> <string>
<align> ::= "align" <equal> <number>
<pattern> ::= "pattern" <equal> <number>
<patterncolor> ::= "patterncolor" <equal> <color>
<frame> ::= "frame" <equal> <color>
<data> ::= "data" <equal> <number> | <string>
<framethick> ::= "framethick"
<wordwrap> ::= "wordwrap"

```

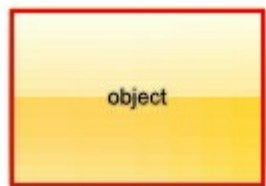
Others like: pic, stretch, hstretch, vstretch, transparent, from, to are reserved for future use only.

Here's a few easy samples:

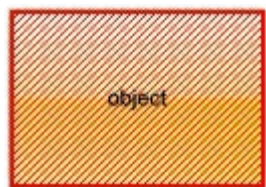
- "[pattern=6]", shows the BDiagonal pattern on the object's background.



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

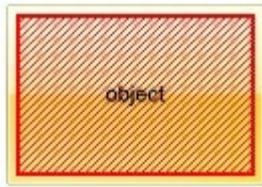


- "[frame=RGB(255,0,0),framethick,pattern=6,patterncolor=RGB(255,0,0)]", draws a red thick-border around the object, with a patter inside.

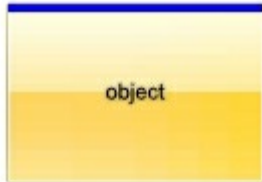


- "[[patterncolor=RGB(255,0,0)]
(none[(4,4,100%-8,100%-8),pattern=0x006,patterncolor=RGB(255,0,0),frame=RGB(255,0,0),framethick])]", draws a red thick-border around the object, with a patter inside, with a 4-pixels wide

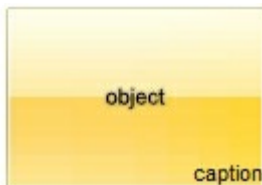
padding:



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



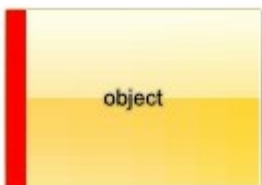
- `"[text=`caption`,align=0x22]"`, shows the caption string aligned to the bottom-right side of the object's background.



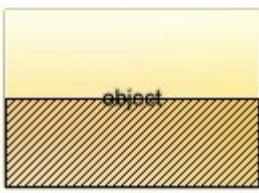
- `"[text=`flag`,align=0x11]"` shows the flag picture and the sweden string aligned to the bottom side of the object.



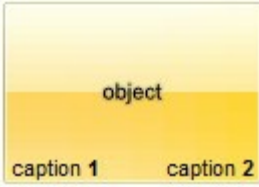
- `"left[10,back=RGB(255,0,0)]"`, draws a red line on the left side of the object's background, of 10-pixels wide.



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



- "root[text=`caption 2`,align=0x22](client[text=`caption 1`,align=0x20])", shows the caption **1** aligned to the bottom-left side, and the caption **2** to the bottom-right side



property Ribbon.ToTemplate ([DefaultTemplate as Variant]) as String

Generates the control's template.

Type	Description
DefaultTemplate as Variant	A String expression that defines the properties/methods to be generated. If missing, the control's automatically generates code for all properties.
String	A String expression that indicates the x--script code.

The ToTemplate property generates x-script from the control's content. The [AllowCopyTemplate](#) property specifies whether the Shift + Ctrl + Alt + Insert sequence copies the control's content to the clipboard, in template form. The [Template](#) property can be used to place the result of ToTemplate property. Use the [ExecuteTemplate](#) property to execute a template script and gets the result.

Here's how the x-script generated could look as:

```
Font
{
    Size = 7.8
}
ShortcutKeyPadding = "3,0,3,0"
Items
{
    Add("",2,10)
    {
        EditWidth = 40
        GroupPopup = 3
        Items
        {
            Add("Item <br> <c> <b> 1",0,20)
            {
                EditWidth = 40
            }
            Add("Item <br> <c> <b> 2",0,30)
            {
                EditWidth = 40
            }
        }
    }
}
```



```

Add("Item <br> <c> <b>3",0,40)
{
    EditWidth = 40
}
}
}
}
Refresh()

```

for ribbon control such as:



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 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 Ribbon.Update ()

Updates the control.

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

The Update method invalidates the entire control, without resizing the elements inside. Use the [Update](#) method to update a particular item. The Update method does not re-computes the required size of the item, so it just validates the item's client area. The [Refresh](#) method refreshes the entire control, including resizing inside elements.

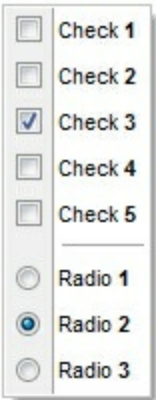
property Ribbon.UseVisualTheme as UIVisualThemeEnum

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 [SelBackColor](#) property specifies the visual appearance of the selected / highlighted item.

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 Ribbon.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 Ribbon.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. Use the [Appearance](#) property to change the menu's frame using an EBN skin object. The [SelBackColor](#) property specifies the visual appearance of the item being selected / highlighted. The [Background](#) property specifies the visual appearance for different parts of the control, including the radio-buttons, check-boxes or separator items.

The following screen shot shows the control's frame using a different EBN file:



ExRibbon events

Tip The /COM object can be placed on a HTML page (with usage of the HTML object tag: <object classid="clsid:...">) using the class identifier: {DDF58CFA-750F-45E0-8A00-CFBE431702E2}. The object's program identifier is: "Exontrol.Ribbon". The /COM object module is: "ExRibbon.dll"

The trial/evaluation version of the control limit firing the events. In other words, using the trial/evaluation version won't fire the event every time. The exRibbon component supports the following events:

Name	Description
CheckItem	Occurs when the user checks the item.
Click	Occurs when the user presses and then releases the left mouse button over the tree control.
DbClick	Occurs when the user dblclk the left mouse button over an object.
EditChange	Occurs when the user alters the item's text box field.
Event	Notifies the application once the control fires an event.
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.
OleEvent	Occurs when an inside ActiveX control fires an event.
SelectItem	Occurs when the user selects the item.
UncheckItem	Occurs when the user unchecks the item.

event CheckItem (Itm as Item)

Occurs when the user checks the item.

Type	Description
Itm as Item	An Item object that indicates the item being checked.

The trial/evaluation version of the control limits firing this event. In other words, using the trial/evaluation version won't fire the event every time.

The CheckItem event notifies your application once the user clicks the item's check-box. The [Check](#) property indicates whether the Item has associated a check box. The [Checked](#) property specifies whether the item is checked or unchecked. Use the [Radio](#) property to display a radio-button on the item. In C++ or VFP, you can use the [Notifier](#) to get notified through the WM_COMMAND message. The [Checked](#) property specifies whether the item is checked or unchecked. The [UncheckedItem](#) event notifies once the user un-checks the item.

Syntax for CheckItem event, **/NET** version, on:

```
C# private void CheckItem(object sender,exontrol.EXRIBBONLib.Item Itm)
{
}
```

```
VB Private Sub CheckItem(ByVal sender As System.Object,ByVal Itm As
exontrol.EXRIBBONLib.Item) Handles CheckItem
End Sub
```

Syntax for CheckItem event, **/COM** version, on:

```
C# private void CheckItem(object sender,
AxEXRIBBONLib._IRibbonEvents_CheckItemEvent e)
{
}
```

```
C++ void OnCheckItem(LPDISPATCH Itm)
{
}
```

```
C++ Builder void __fastcall CheckItem(TObject *Sender,Exribbonlib_tlb::IItem *Itm)
{
}
```



```
}
```

Delphi

```
procedure CheckItem(ASender: TObject; Itm : IItem);  
begin  
end;
```

Delphi 8
(.NET
only)

```
procedure CheckItem(sender: System.Object; e:  
AxEXRIBBONLib._IRibbonEvents_CheckItemEvent);  
begin  
end;
```

Powe...

```
begin event CheckItem(oleobject Itm)  
end event CheckItem
```

VB.NET

```
Private Sub CheckItem(ByVal sender As System.Object, ByVal e As  
AxEXRIBBONLib._IRibbonEvents_CheckItemEvent) Handles CheckItem  
End Sub
```

VB6

```
Private Sub CheckItem(ByVal Itm As EXRIBBONLibCtl.IItem)  
End Sub
```

VBA

```
Private Sub CheckItem(ByVal Itm As Object)  
End Sub
```

VFP

```
LPARAMETERS Itm
```

Xbas...

```
PROCEDURE OnCheckItem(oRibbon,Itm)  
RETURN
```

Syntax for CheckItem event, **/COM** version (others), on:

Java...

```
<SCRIPT EVENT="CheckItem(Itm)" LANGUAGE="JScript">  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function CheckItem(Itm)  
End Function
```

```
</SCRIPT>
```

Visual
Data...

```
Procedure OnComCheckItem Variant lltm  
    Forward Send OnComCheckItem lltm  
End_Procedure
```

Visual
Objects

```
METHOD OCX_CheckItem(ltm) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_CheckItem(COM _ltm)  
{  
}
```

XBasic

```
function CheckItem as v (ltm as OLE::Exontrol.Ribbon.1::Item)  
end function
```

dBASE

```
function nativeObject_CheckItem(ltm)  
return
```

The following samples show how you can get notified once the user checks an item:

VBA (MS Access, Excell...)

' CheckItem event - Occurs when the user checks the item.

```
Private Sub Ribbon1_CheckItem(ByVal ltm As Object)  
    With Ribbon1  
        Debug.Print( "CheckItem event on ltm object" )  
    End With  
End Sub
```

```
With Ribbon1  
    With .Items  
        .Add("Item").Check = True  
        .Add("Item").Check = True  
    End With  
    .Refresh  
End With
```

VB6

' **CheckItem event - Occurs when the user checks the item.**

```
Private Sub Ribbon1_CheckItem(ByVal Itm As EXRIBBONLibCtl.Item)
    With Ribbon1
        Debug.Print( "CheckItem event on Itm object" )
    End With
End Sub

With Ribbon1
    With .Items
        .Add("Item").Check = True
        .Add("Item").Check = True
    End With
    .Refresh
End With
```

VB.NET

' **CheckItem event - Occurs when the user checks the item.**

```
Private Sub Exribbon1_CheckItem(ByVal sender As System.Object, ByVal Itm As
exontrol.EXRIBBONLib.Item) Handles Exribbon1.CheckItem
    With Exribbon1
        Debug.Print( "CheckItem event on Itm object" )
    End With
End Sub

With Exribbon1
    With .Items
        .Add("Item").Check = True
        .Add("Item").Check = True
    End With
    .Refresh()
End With
```

VB.NET for /COM

' **CheckItem event - Occurs when the user checks the item.**

```

Private Sub AxRibbon1_CheckItem(ByVal sender As System.Object, ByVal e As
AxEXRIBBONLib._IRibbonEvents_CheckItemEvent) Handles AxRibbon1.CheckItem
    With AxRibbon1
        Debug.Print( "CheckItem event on Itm object" )
    End With
End Sub

With AxRibbon1
    With .Items
        .Add("Item").Check = True
        .Add("Item").Check = True
    End With
    .Refresh()
End With

```

C++

// CheckItem event - Occurs when the user checks the item.

```

void OnCheckItemRibbon1(LPDISPATCH Itm)
{
    /*
        Copy and paste the following directives to your header file as
        it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control
Library'
        #import <ExRibbon.dll>
        using namespace EXRIBBONLib;
    */
    EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
    OutputDebugStringW( L"CheckItem event on Itm object" );
}

```

```

EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
var_Items->Add(L"Item",vtMissing,vtMissing)->PutCheck(VARIANT_TRUE);
var_Items->Add(L"Item",vtMissing,vtMissing)->PutCheck(VARIANT_TRUE);

```

```
spRibbon1->Refresh();
```

C++ Builder

// CheckItem event - Occurs when the user checks the item.

```
void __fastcall TForm1::Ribbon1CheckItem(TObject *Sender,Exribbonlib_tlb::Item
*Itm)
{
    OutputDebugString( L"CheckItem event on Itm object" );
}
```

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
var_Items->Add(L"Item",TNoParam(),TNoParam())->Check = true;
var_Items->Add(L"Item",TNoParam(),TNoParam())->Check = true;
Ribbon1->Refresh();
```

C#

// CheckItem event - Occurs when the user checks the item.

```
private void exribbon1_CheckItem(object sender,exontrol.EXRIBBONLib.Item Itm)
{
    System.Diagnostics.Debug.Print( "CheckItem event on Itm object" );
}
```

//this.exribbon1.CheckItem += new

exontrol.EXRIBBONLib.exg2antt.CheckItemEventHandler(this.exribbon1_CheckItem)

```
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
var_Items.Add("Item",null,null).Check = true;
var_Items.Add("Item",null,null).Check = true;
exribbon1.Refresh();
```

JSript/JavaScript

```
<BODY onload='Init()>
```

```

<SCRIPT FOR="Ribbon1" EVENT="CheckItem(Itm)" LANGUAGE="JScript">
    alert( "CheckItem event on Itm object" );
</SCRIPT>

<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var_Items.Add("Item",null,null).Check = true;
    var_Items.Add("Item",null,null).Check = true;
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()'>
<SCRIPT LANGUAGE="VBScript">
Function Ribbon1_CheckItem(Itm)
    With Ribbon1
        alert( "CheckItem event on Itm object" )
    End With
End Function
</SCRIPT>

<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items

```

```

        .Add("Item").Check = True
        .Add("Item").Check = True
    End With
    .Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

// CheckItem event - Occurs when the user checks the item.

```

private void axRibbon1_CheckItem(object sender,
AxEXRIBBONLib._IRibbonEvents_CheckItemEvent e)
{
    System.Diagnostics.Debug.Print( "CheckItem event on Itm object" );
}

```

//this.axRibbon1.CheckItem += new

AxEXRIBBONLib._IRibbonEvents_CheckItemEventHandler(this.axRibbon1_Check

```

EXRIBBONLib.Items var_Items = axRibbon1.Items;
    var_Items.Add("Item",null,null).Check = true;
    var_Items.Add("Item",null,null).Check = true;
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

// CheckItem event - Occurs when the user checks the item.

```

void onEvent_CheckItem(COM _Itm)
{
    ;
    print( "CheckItem event on Itm object" );
}

public void init()

```

```

{
  COM com_Item,com_Items;
  anytype var_Item,var_Items;
  ;

  super();

  var_Items = exribbon1.Items(); com_Items = var_Items;
  var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
  com_Item.Check(true);
  var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
  com_Item.Check(true);
  exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

// CheckItem event - Occurs when the user checks the item.

```

procedure TForm1.AxRibbon1_CheckItem(sender: System.Object; e:
AxEXRIBBONLib._IRibbonEvents_CheckItemEvent);
begin
  with AxRibbon1 do
  begin
    OutputDebugString( 'CheckItem event on itm object' );
  end
end;

with AxRibbon1 do
begin
  with Items do
  begin
    Add('Item',Nil,Nil).Check := True;
    Add('Item',Nil,Nil).Check := True;
  end;
  Refresh();
end;

```



```
end
```

Delphi (standard)

// CheckItem event - Occurs when the user checks the item.

```
procedure TForm1.Ribbon1CheckItem(ASender: TObject; Itm : IItem);
begin
  with Ribbon1 do
  begin
    OutputDebugString( 'CheckItem event on Itm object' );
  end
end;

with Ribbon1 do
begin
  with Items do
  begin
    Add('Item',Null,Null).Check := True;
    Add('Item',Null,Null).Check := True;
  end;
  Refresh();
end
```

VFP

*** **CheckItem** event - Occurs when the user checks the item. ***

```
LPARAMETERS Itm
  with thisform.Ribbon1
    DEBUGOUT( "CheckItem event on Itm object" )
  endwith

with thisform.Ribbon1
  with .Items
    .Add("Item").Check = .T.
    .Add("Item").Check = .T.
  endwith
  .Refresh
endwith
```

```
/*  
with (this.ACTIVEX1.nativeObject)  
    CheckItem = class::nativeObject_CheckItem  
endwith  
*/  
// Occurs when the user checks the item.  
function nativeObject_CheckItem(ltm)  
    local oRibbon  
    oRibbon = form.Activex1.nativeObject  
    ? "CheckItem event on ltm object"  
return  
  
local oRibbon,var_Item,var_Item1,var_Items  
  
oRibbon = form.Activex1.nativeObject  
var_Items = oRibbon.Items  
    // var_Items.Add("Item").Check = true  
var_Item = var_Items.Add("Item")  
with (oRibbon)  
    TemplateDef = [Dim var_Item]  
    TemplateDef = var_Item  
    Template = [var_Item.Check = true]  
endwith  
    // var_Items.Add("Item").Check = true  
var_Item1 = var_Items.Add("Item")  
with (oRibbon)  
    TemplateDef = [Dim var_Item1]  
    TemplateDef = var_Item1  
    Template = [var_Item1.Check = true]  
endwith  
oRibbon.Refresh()
```

```
function CheckItem as v (itm as OLE::Exontrol.Ribbon.1::Item)
```

```
    Dim oRibbon as P
```

```
    oRibbon = topparent:CONTROL_ACTIVEX1.activex
```

```
    ? "CheckItem event on Itm object"
```

```
end function
```

```
Dim oRibbon as P
```

```
Dim var_Item as P
```

```
Dim var_Item1 as P
```

```
Dim var_Items as P
```

```
oRibbon = topparent:CONTROL_ACTIVEX1.activex
```

```
var_Items = oRibbon.Items
```

```
    ' var_Items.Add("Item").Check = .t.
```

```
    var_Item = var_Items.Add("Item")
```

```
    oRibbon.TemplateDef = "Dim var_Item"
```

```
    oRibbon.TemplateDef = var_Item
```

```
    oRibbon.Template = "var_Item.Check = True"
```

```
    ' var_Items.Add("Item").Check = .t.
```

```
    var_Item1 = var_Items.Add("Item")
```

```
    oRibbon.TemplateDef = "Dim var_Item1"
```

```
    oRibbon.TemplateDef = var_Item1
```

```
    oRibbon.Template = "var_Item1.Check = True"
```

```
oRibbon.Refresh()
```

Visual Objects

```
METHOD OCX_Exontrol1CheckItem(Itm) CLASS MainDialog
```

```
    // CheckItem event - Occurs when the user checks the item.
```

```
    OutputDebugString(String2Psz( "CheckItem event on Itm object" ))
```

```
RETURN NIL
```

```
local var_Items as IItems
```

```

var_Items := oDCOCX_Exontrol1:Items
  var_Items.Add("Item",nil,nil):Check := true
  var_Items.Add("Item",nil,nil):Check := true
oDCOCX_Exontrol1:Refresh()

```

PowerBuilder

```

/*begin event CheckItem(oleobject Itm) - Occurs when the user checks the item.*/
/*
  OleObject oRibbon
  oRibbon = ole_1.Object
  MessageBox("Information",string( "CheckItem event on Itm object" ))
*/
/*end event CheckItem*/

```

```
OleObject oRibbon,var_Items
```

```

oRibbon = ole_1.Object
var_Items = oRibbon.Items
  var_Items.Add("Item").Check = true
  var_Items.Add("Item").Check = true
oRibbon.Refresh()

```

Visual DataFlex

// Occurs when the user checks the item.

```

Procedure OnComCheckItem Variant llltm
  Forward Send OnComCheckItem llltm
  Showln "CheckItem event on Itm object"
End_Procedure

```

```

Procedure OnCreate
  Forward Send OnCreate
  Variant voltems
  Get ComItems to voltems
  Handle holtems

```

```

Get Create (RefClass(cComltems)) to holtems
Set pvComObject of holtems to voltems
Variant voltem
Get ComAdd of holtems "Item" Nothing Nothing to voltem
Handle holtem
Get Create (RefClass(cComltem)) to holtem
Set pvComObject of holtem to voltem
    Set ComCheck of holtem to True
Send Destroy to holtem
Variant voltem1
Get ComAdd of holtems "Item" Nothing Nothing to voltem1
Handle holtem1
Get Create (RefClass(cComltem)) to holtem1
Set pvComObject of holtem1 to voltem1
    Set ComCheck of holtem1 to True
Send Destroy to holtem1
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

PROCEDURE OnCheckItem(oRibbon,Itm)
    DevOut( "CheckItem event on Itm object" )
RETURN

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.

```

```

oForm:create( ,, {100,100}, {640,480},,, .F. )
oForm:close := {|| PostAppEvent( xbeP_Quit )}

oRibbon := XbpActiveXControl():new( oForm:drawingArea )
oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
oRibbon:create(,, {10,60},{610,370} )

    oRibbon:CheckItem := {|Itm| OnCheckItem(oRibbon,Itm)} /*Occurs when the
user checks the item.*/

    oltems := oRibbon:Items()
        oltems:Add("Item"):Check := .T.
        oltems:Add("Item"):Check := .T.
    oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

event Click ()

Occurs when the user presses and then releases the left mouse button over the tree control.

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

event DbtClick (Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user dblclk the left mouse button over an object.

Type	Description
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates

The DbtClick event is fired when user double clicks the control. The [ItemFromPoint\(-1,-1\)](#) property gets the item from the cursor.

Syntax for DbtClick event, **/NET** version, on:

C#private void DbtClick(object sender,short Shift,int X,int Y)
{
}

VBPrivate Sub DbtClick(ByVal sender As System.Object,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles DbtClick
End Sub

Syntax for DbtClick event, **/COM** version, on:

C#private void DbtClick(object sender, AxEXRibbonLib._IRibbonEvents_DbtClickEvent e)
{
}

C++void OnDbtClick(short Shift,long X,long Y)
{
}

C++ Buildervoid __fastcall DbtClick(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:  
AxEXRibbonLib._IRibbonEvents_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  
AxEXRibbonLib._IRibbonEvents_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(oRibbon,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 DbClick(Shift,X,Y)  
End Function  
</SCRIPT>
```

Visual
Data...

```
Procedure OnComDbClick Short IIShift OLE_XPOS_PIXELS IIX OLE_YPOS_PIXELS  
IYY  
    Forward Send OnComDbClick IIShift IIX IYY  
End_Procedure
```

Visual
Objects

```
METHOD OCX_DbClick(Shift,X,Y) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_DbClick(int _Shift,int _X,int _Y)  
{  
}  
}
```

XBasic

```
function DbClick as v (Shift as N,X as  
OLE::Exontrol.Ribbon.1::OLE_XPOS_PIXELS,Y as  
OLE::Exontrol.Ribbon.1::OLE_YPOS_PIXELS)  
end function
```

dBASE

```
function nativeObject_DbClick(Shift,X,Y)  
return
```

event EditChange (Itm as Item)

Occurs when the user alters the item's text box/editor field.

Type	Description
Itm as Item	An Item object that contains a editor inside.

The trial/evaluation version of the control limits firing this event. In other words, using the trial/evaluation version won't fire the event every time.

The EditChange event notifies your application once the user alters the item's editor value. The [EditCaption](#) property specifies the caption of the editor being altered. Use the [AllowEdit](#) property to add a editor inside the item, so the user can type any characters inside. The [EditWidth](#) property specifies the width of the editor inside the item. The [EditBorder](#) property specifies the border to be shown around the item's text box. You can use the [Get](#) method to collect all items of Edit type. In C++ or VFP, you can use the [Notifier](#) to get notified through the WM_COMMAND message.

Syntax for EditChange event, **/NET** version, on:

```
C# private void EditChange(object sender,exontrol.EXRIBBONLib.Item Itm)
{
}
```

```
VB Private Sub EditChange(ByVal sender As System.Object,ByVal Itm As
exontrol.EXRIBBONLib.Item) Handles EditChange
End Sub
```

Syntax for EditChange event, **/COM** version, on:

```
C# private void EditChange(object sender,
AxEXRIBBONLib._IRibbonEvents_EditChangeEvent e)
{
}
```

```
C++ void OnEditChange(LPDISPATCH Itm)
{
}
```

```
C++ Builder void __fastcall EditChange(TObject *Sender,Exribbonlib_tlb::IItem *Itm)
{
```

```
}
```

Delphi

```
procedure EditChange(ASender: TObject; Itm : IItem);  
begin  
end;
```

Delphi 8
(.NET
only)

```
procedure EditChange(sender: System.Object; e:  
AxEXRIBBONLib._IRibbonEvents_EditChangeEvent);  
begin  
end;
```

Powe...

```
begin event EditChange(oleobject Itm)  
end event EditChange
```

VB.NET

```
Private Sub EditChange(ByVal sender As System.Object, ByVal e As  
AxEXRIBBONLib._IRibbonEvents_EditChangeEvent) Handles EditChange  
End Sub
```

VB6

```
Private Sub EditChange(ByVal Itm As EXRIBBONLibCtl.IItem)  
End Sub
```

VBA

```
Private Sub EditChange(ByVal Itm As Object)  
End Sub
```

VFP

```
LPARAMETERS Itm
```

Xbas...

```
PROCEDURE OnEditChange(oRibbon,Itm)  
RETURN
```

Syntax for EditChange event, **/COM** version (others), on:

Java...

```
<SCRIPT EVENT="EditChange(Itm)" LANGUAGE="JScript">  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function EditChange(Itm)  
End Function
```

</SCRIPT>

Visual
Data...

```
Procedure OnComEditChange Variant lItm  
    Forward Send OnComEditChange lItm  
End_Procedure
```

Visual
Objects

```
METHOD OCX_EditChange(ltm) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_EditChange(COM _ltm)  
{  
}
```

XBasic

```
function EditChange as v (ltm as OLE::Exontrol.Ribbon.1::Item)  
end function
```

dBASE

```
function nativeObject_EditChange(ltm)  
return
```

The following samples show how you can get notified once the user edits the field, or change the slider value:

VBA (MS Access, Excell...)

' EditChange event - Occurs when the user alters the item's text box field.

```
Private Sub Ribbon1_EditChange(ByVal ltm As Object)  
    With Ribbon1  
        Debug.Print( "EditChange event on ltm object" )  
    End With  
End Sub
```

```
With Ribbon1  
    With .Items  
        With .Add("Item")  
            .AllowEdit = 3  
            .EditWidth = 128  
            .EditBorder = 0  
            .EditValue = 25
```

```
End With
End With
.Refresh
End With
```

VB6

' EditChange event - Occurs when the user alters the item's text box field.

```
Private Sub Ribbon1_EditChange(ByVal Itm As EXRIBBONLibCtl.IItem)
```

```
    With Ribbon1
```

```
        Debug.Print( "EditChange event on Itm object" )
```

```
    End With
```

```
End Sub
```

```
With Ribbon1
```

```
    With .Items
```

```
        With .Add("Item")
```

```
            .AllowEdit = exItemEditSlider
```

```
            .EditWidth = 128
```

```
            .EditBorder = exEditBorderNone
```

```
            .EditValue = 25
```

```
        End With
```

```
    End With
```

```
    .Refresh
```

```
End With
```

VB.NET

' EditChange event - Occurs when the user alters the item's text box field.

```
Private Sub Exribbon1_EditChange(ByVal sender As System.Object, ByVal Itm As  
exontrol.EXRIBBONLib.Item) Handles Exribbon1.EditChange
```

```
    With Exribbon1
```

```
        Debug.Print( "EditChange event on Itm object" )
```

```
    End With
```

```
End Sub
```

```
With Exribbon1
```

```
    With .Items
```

```

With .Add("Item")
    .AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditSlider
    .EditWidth = 128
    .EditBorder = exontrol.EXRIBBONLib.EditBorderEnum.exEditBorderNone
    .EditValue = 25
End With
End With
.Refresh()
End With

```

VB.NET for /COM

' EditChange event - Occurs when the user alters the item's text box field.

```

Private Sub AxRibbon1_EditChange(ByVal sender As System.Object, ByVal e As
AxEXRIBBONLib._IRibbonEvents_EditChangeEvent) Handles AxRibbon1.EditChange
    With AxRibbon1
        Debug.Print( "EditChange event on Itm object" )
    End With
End Sub

```

```

With AxRibbon1
    With .Items
        With .Add("Item")
            .AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditSlider
            .EditWidth = 128
            .EditBorder = EXRIBBONLib.EditBorderEnum.exEditBorderNone
            .EditValue = 25
        End With
    End With
    .Refresh()
End With

```

C++

// EditChange event - Occurs when the user alters the item's text box field.

```

void OnEditChangeRibbon1(LPDISPATCH Itm)
{
    /*

```

Copy and paste the following directives to your header file as it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control Library'

```
#import <ExRibbon.dll>
using namespace EXRIBBONLib;
*/
EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
    OutputDebugStringW( L"EditChange event on Itm object" );
}

EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
    EXRIBBONLib::IItemPtr var_Item = var_Items->Add(L"Item",vtMissing,vtMissing);
        var_Item->PutAllowEdit(EXRIBBONLib::exItemEditSlider);
        var_Item->PutEditWidth(128);
        var_Item->PutEditBorder(EXRIBBONLib::exEditBorderNone);
        var_Item->PutEditValue(long(25));
spRibbon1->Refresh();
```

C++ Builder

// EditChange event - Occurs when the user alters the item's text box field.

```
void __fastcall TForm1::Ribbon1EditChange(TObject *Sender,Exribbonlib_tlb::IItem
*IItem)
{
    OutputDebugString( L"EditChange event on Itm object" );
}
```

```
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
    Exribbonlib_tlb::IItemPtr var_Item = var_Items-
>Add(L"Item",TNoParam(),TNoParam());
        var_Item->AllowEdit = Exribbonlib_tlb::AllowEditEnum::exItemEditSlider;
        var_Item->EditWidth = 128;
        var_Item->EditBorder = Exribbonlib_tlb::EditBorderEnum::exEditBorderNone;
```



```
var_Item->set_EditValue(TVariant(25));  
Ribbon1->Refresh();
```

C#

```
// EditChange event - Occurs when the user alters the item's text box field.  
private void exribbon1_EditChange(object sender,exontrol.EXRIBBONLib.Item Itm)  
{  
    System.Diagnostics.Debug.Print( "EditChange event on Itm object" );  
}  
  
//this.exribbon1.EditChange += new  
exontrol.EXRIBBONLib.exg2antt.EditChangeEventHandler(this.exribbon1_EditCr  
  
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
exontrol.EXRIBBONLib.Item var_Item = var_Items.Add("Item",null,null);  
var_Item.AllowEdit = exontrol.EXRIBBONLib.AllowEditEnum.exItemEditSlider;  
var_Item.EditWidth = 128;  
var_Item.EditBorder = exontrol.EXRIBBONLib.EditBorderEnum.exEditBorderNone;  
var_Item.EditValue = 25;  
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>  
<SCRIPT FOR="Ribbon1" EVENT="EditChange(Itm)" LANGUAGE="JScript">  
    alert( "EditChange event on Itm object" );  
</SCRIPT>  
  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    var var_Items = Ribbon1.Items;
```

```

    var var_Item = var_Items.Add("Item",null,null);
    var_Item.AllowEdit = 3;
    var_Item.EditWidth = 128;
    var_Item.EditBorder = 0;
    var_Item.EditValue = 25;
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>

```

VBScript

```

<BODY onload='Init()>
<SCRIPT LANGUAGE="VBScript">
Function Ribbon1_EditChange(ltm)
    With Ribbon1
        alert( "EditChange event on ltm object" )
    End With
End Function
</SCRIPT>

<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
        With .Items
            With .Add("Item")
                .AllowEdit = 3
                .EditWidth = 128
                .EditBorder = 0
                .EditValue = 25
            End With
        End With
    End With
    .Refresh

```

```
End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
// EditChange event - Occurs when the user alters the item's text box field.
private void axRibbon1_EditChange(object sender,
AxEXRIBBONLib._IRibbonEvents_EditChangeEvent e)
{
    System.Diagnostics.Debug.Print( "EditChange event on Itm object" );
}
//this.axRibbon1.EditChange += new
AxEXRIBBONLib._IRibbonEvents_EditChangeEventHandler(this.axRibbon1_EditC

EXRIBBONLib.Items var_Items = axRibbon1.Items;
EXRIBBONLib.Item var_Item = var_Items.Add("Item",null,null);
    var_Item.AllowEdit = EXRIBBONLib.AllowEditEnum.exItemEditSlider;
    var_Item.EditWidth = 128;
    var_Item.EditBorder = EXRIBBONLib.EditBorderEnum.exEditBorderNone;
    var_Item.EditValue = 25;
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
// EditChange event - Occurs when the user alters the item's text box field.
void onEvent_EditChange(COM _Itm)
{
    ;
    print( "EditChange event on Itm object" );
}

public void init()
{
```

```

COM com_Item,com_Items;
anytype var_Item,var_Items;
;

super();

var_Items = exribbon1.Items(); com_Items = var_Items;
var_Item = com_Items.Add("Item"); com_Item = var_Item;
com_Item.AllowEdit(3/*exItemEditSlider*/);
com_Item.EditWidth(128);
com_Item.EditBorder(0/*exEditBorderNone*/);
com_Item.EditValue(COMVariant::createFromInt(25));
exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

// EditChange event - Occurs when the user alters the item's text box field.

```

procedure TForm1.AxRibbon1_EditChange(sender: System.Object; e:
AxEXRIBBONLib._IRibbonEvents_EditChangeEvent);

```

```

begin
  with AxRibbon1 do
    begin
      OutputDebugString( 'EditChange event on itm object' );
    end
  end;

```

```

with AxRibbon1 do
begin
  with Items do
    begin
      with Add('Item',Nil,Nil) do
        begin
          AllowEdit := EXRIBBONLib.AllowEditEnum.exItemEditSlider;
          EditWidth := 128;
          EditBorder := EXRIBBONLib.EditBorderEnum.exEditBorderNone;
         .EditValue := TObject(25);

```

```
end;  
end;  
Refresh();  
end
```

Delphi (standard)

// EditChange event - Occurs when the user alters the item's text box field.

```
procedure TForm1.Ribbon1EditChange(ASender: TObject; Itm : IItem);  
begin  
  with Ribbon1 do  
  begin  
    OutputDebugString( 'EditChange event on Itm object' );  
  end  
end;  
  
with Ribbon1 do  
begin  
  with Items do  
  begin  
    with Add('Item',Null,Null) do  
    begin  
      AllowEdit := EXRIBBONLib_TLB.exItemEditSlider;  
      EditWidth := 128;  
      EditBorder := EXRIBBONLib_TLB.exEditBorderNone;  
      EditValue := OleVariant(25);  
    end;  
  end;  
end;  
Refresh();  
end
```

VFP

***** EditChange event - Occurs when the user alters the item's text box field. *****

```
LPARAMETERS Itm  
  with thisform.Ribbon1  
    DEBUGOUT( "EditChange event on Itm object" )  
  endwith
```

```

with thisform.Ribbon1
  with .Items
    with .Add("Item")
      .AllowEdit = 3
      .EditWidth = 128
      .EditBorder = 0
      .EditValue = 25
    endwhile
  endwhile
.Refresh
endwith

```

dBASE Plus

```

/*
with (this.ACTIVEX1.nativeObject)
  EditChange = class::nativeObject_EditChange
endwith
*/
// Occurs when the user alters the item's text box field.
function nativeObject_EditChange(ltm)
  local oRibbon
  oRibbon = form.Activex1.nativeObject
  ? "EditChange event on ltm object"
return

local oRibbon,var_Item,var_Items

oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
var_Item = var_Items.Add("Item")
  var_Item.AllowEdit = 3
  var_Item.EditWidth = 128
  var_Item.EditBorder = 0
  var_Item.EditValue = 25
oRibbon.Refresh()

```

XBasic (Alpha Five)

' Occurs when the user alters the item's text box field.

```
function EditChange as v (Itm as OLE::Exontrol.Ribbon.1::Item)
```

```
    Dim oRibbon as P
```

```
    oRibbon = topparent:CONTROL_ACTIVEX1.activex
```

```
    ? "EditChange event on Itm object"
```

```
end function
```

```
Dim oRibbon as P
```

```
Dim var_Item as P
```

```
Dim var_Items as P
```

```
oRibbon = topparent:CONTROL_ACTIVEX1.activex
```

```
var_Items = oRibbon.Items
```

```
    var_Item = var_Items.Add("Item")
```

```
        var_Item.AllowEdit = 3
```

```
        var_Item.EditWidth = 128
```

```
        var_Item.EditBorder = 0
```

```
        var_Item.EditValue = 25
```

```
oRibbon.Refresh()
```

Visual Objects

```
METHOD OCX_Exontrol1EditChange(Itm) CLASS MainDialog
```

```
    // EditChange event - Occurs when the user alters the item's text box field.
```

```
    OutputDebugString(String2Psz( "EditChange event on Itm object" ))
```

```
RETURN NIL
```

```
local var_Item as Item
```

```
local var_Items as Items
```

```
var_Items := oDCOCX_Exontrol1:Items
```

```
    var_Item := var_Items:Add("Item",nil,nil)
```

```
        var_Item.AllowEdit := exItemEditSlider
```

```
var_Item:EditWidth := 128
var_Item:EditBorder := exEditBorderNone
var_Item:EditValue := 25
oDCOCX_Exontrol1:Refresh()
```

PowerBuilder

```
/*begin event EditChange(oleobject ltm) - Occurs when the user alters the item's text
box field.*/
/*
    OleObject oRibbon
    oRibbon = ole_1.Object
    MessageBox("Information",string( "EditChange event on ltm object" ))
*/
/*end event EditChange*/
```

```
OleObject oRibbon,var_Item,var_Items
```

```
oRibbon = ole_1.Object
var_Items = oRibbon.Items
var_Item = var_Items.Add("Item")
var_Item.AllowEdit = 3
var_Item.EditWidth = 128
var_Item.EditBorder = 0
var_Item.EditValue = 25
oRibbon.Refresh()
```

Visual DataFlex

// Occurs when the user alters the item's text box field.

```
Procedure OnComEditChange Variant llitm
    Forward Send OnComEditChange llitm
    Showln "EditChange event on ltm object"
End_Procedure
```

```
Procedure OnCreate
```



```

Forward Send OnCreate
Variant voltems
Get Comltems to voltems
Handle holtems
Get Create (RefClass(cComltems)) to holtems
Set pvComObject of holtems to voltems
    Variant voltem
    Get ComAdd of holtems "Item" Nothing Nothing to voltem
    Handle holtem
    Get Create (RefClass(cComltem)) to holtem
    Set pvComObject of holtem to voltem
        Set ComAllowEdit of holtem to OLEexltemEditSlider
        Set ComEditWidth of holtem to 128
        Set ComEditBorder of holtem to OLEexEditBorderNone
        Set ComEditValue of holtem to 25
    Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

PROCEDURE OnEditChange(oRibbon,Itm)
    DevOut( "EditChange event on Itm object" )
RETURN

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltem
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )

```

```

oForm:drawingArea:clipChildren := .T.
oForm:create( ,, {100,100}, {640,480},,, .F. )
oForm:close := {|| PostAppEvent( xbeP_Quit )}

oRibbon := XbpActiveXControl():new( oForm:drawingArea )
oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
oRibbon:create(,, {10,60},{610,370} )

    oRibbon:EditChange := {||itm| OnEditChange(oRibbon,itm)} /*Occurs when the
user alters the item's text box field.*/

oltems := oRibbon:Items()
    oltem := oltems:Add("Item")
        oltem:AllowEdit := 3/*exItemEditSlider*/
        oltem:EditWidth := 128
        oltem:EditBorder := 0/*exEditBorderNone*/
        oltem:EditValue := 25
oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
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 trial/evaluation version of the control limits firing this event. In other words, using the trial/evaluation version won't fire the event every time.

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 exribbon1.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 be 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, AxEXRIBBONLib._IRibbonEvents_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: AxEXRIBBONLib._IRibbonEvents_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  
AxEXRIBBONLib._IRibbonEvents_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(oRibbon,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 lEventID  
Forward Send OnComEvent lEventID
```

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 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 [DbClick](#) events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. The [ItemFromPoint\(-1,-1\)](#) property gets the item from the cursor.

Syntax for MouseDown event, **/NET** version, on:

```
C# private void MouseDownEvent(object sender,short Button,short Shift,int X,int Y)
{
}
```

```
VB Private Sub MouseDownEvent(ByVal sender As System.Object,ByVal Button As Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles MouseDownEvent
End Sub
```

Syntax for MouseDown event, **/COM** version, on:

```
C# private void MouseDownEvent(object sender,
AxEXRibbonLib._IRibbonEvents_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: AxEXRibbonLib._IRibbonEvents_MouseDownEvent);
begin
end;
```

PowerBuilder

```
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 AxEXRibbonLib._IRibbonEvents_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(oRibbon,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  
IIX OLE_YPOS_PIXELS  IY  
    Forward Send OnComMouseDown llButton llShift IIX IY  
End_Procedure
```

Visual
Objects

```
METHOD OCX_MouseDown(Button,Shift,X,Y) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_MouseDown(int  _Button,int  _Shift,int  _X,int  _Y)  
{  
}
```

XBasic

```
function MouseDown as v (Button as N,Shift as N,X as  
OLE::Exontrol.Ribbon.1::OLE_XPOS_PIXELS,Y as  
OLE::Exontrol.Ribbon.1::OLE_YPOS_PIXELS)  
end function
```

dBASE

```
function nativeObject_MouseDown(Button,Shift,X,Y)  
return
```

event MouseEventArgs (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user moves the mouse.

Type	Description
Button as Integer	An integer that corresponds to the state of the mouse buttons in which a bit is set if the button is down. Gets which mouse button was pressed as 1 for Left Mouse Button, 2 for Right Mouse Button and 4 for Middle Mouse Button.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates.

The MouseEventArgs event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseEventArgs event whenever the mouse position is within its borders. The [ItemFromPoint\(-1,-1\)](#) property gets the item from the cursor.

Syntax for MouseEventArgs event, **/NET** version, on:

C#

```
private void MouseEventArgs(object sender,short  Button,short  Shift,int  X,int Y)
{
}
```

VB

```
Private Sub MouseEventArgs(ByVal sender As System.Object,ByVal Button As Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles MouseEventArgs
End Sub
```

Syntax for MouseEventArgs event, **/COM** version, on:

C#

```
private void MouseEventArgs(object sender,
AxEXRibbonLib._IRibbonEvents_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:  
AxEXRibbonLib._IRibbonEvents_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  
AxEXRibbonLib._IRibbonEvents_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(oRibbon,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
IIX OLE_YPOS_PIXELS  ILY
    Forward Send OnComMouseMove llButton llShift IIX ILY
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.Ribbon.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Ribbon.1::OLE_YPOS_PIXELS)
end function
```

dBASE

```
function nativeObject_MouseMove(Button,Shift,X,Y)
return
```

event MouseUp (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Occurs when the user releases a mouse button.

Type	Description
Button as Integer	An integer that identifies the button that was pressed to cause the event.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates.

Use a [MouseDown](#) or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the [Click](#) and [DbClick](#) events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. The [ItemFromPoint\(-1,-1\)](#) property gets the item from the cursor.

Syntax for MouseUp event, **/NET** version, on:

```
C# private void MouseUpEvent(object sender,short  Button,short  Shift,int  X,int  Y)
{
}

VB Private Sub MouseUpEvent(ByVal sender As System.Object,ByVal Button As Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles MouseUpEvent
End Sub
```

Syntax for MouseUp event, **/COM** version, on:

```
C# private void MouseUpEvent(object sender,
AxEXRibbonLib._IRibbonEvents_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: AxEXRibbonLib._IRibbonEvents_MouseUpEvent);
begin
end;

PowerBuilder 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 AxEXRibbonLib._IRibbonEvents_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

Xbase++ PROCEDURE OnMouseUp(oRibbon,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 IButton Short IShift OLE_XPOS_PIXELS IIX
OLE_YPOS_PIXELS IY
 Forward Send OnComMouseUp IButton IShift IIX IY
End_Procedure

Visual Objects METHOD OCX_MouseUp(Button,Shift,X,Y) CLASS MainDialog
RETURN NIL

X++ void onEvent_MouseUp(int _Button,int _Shift,int _X,int _Y)
{
}

XBasic function MouseUp as v (Button as N,Shift as N,X as
OLE::Exontrol.Ribbon.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Ribbon.1::OLE_YPOS_PIXELS)
end function

dBASE function nativeObject_MouseUp(Button,Shift,X,Y)
return

event OleEvent (Itm as Item, Ev as OleEvent)

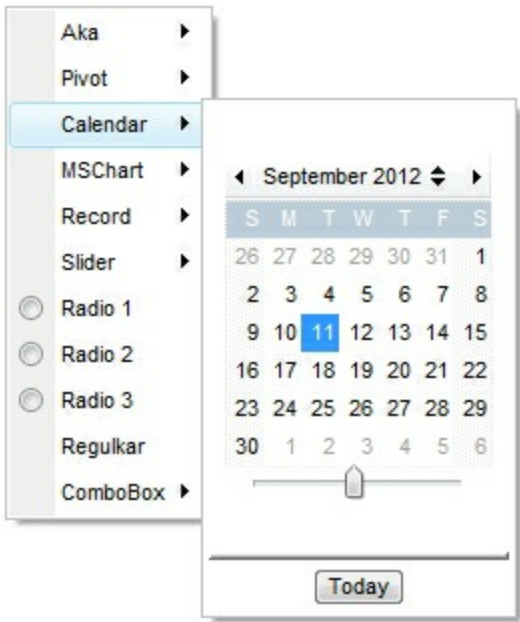
Occurs when an inside ActiveX control fires an event.

Type	Description
Itm as Item	An Item object that contains the sub-control.
Ev as OleEvent	An OleEvent object that holds information about the fired event.

The trial/evaluation version of the control limits firing this event. In other words, using the trial/evaluation version won't fire the event every time.

The eXRibbon component may include sub-menus that displays any ActiveX / NET Component. The inside COM/ActiveX control fires its events through the ExRibbon's OleEvent event. Use the [ItemTypeEnum.SubControl](#) to add an item that hosts an ActiveX inside. Use the [SubControl](#) property to access the properties to create the inside ActiveX control.

The following screen shot displays an item with an [ExCalendar](#) inside:



Syntax for OleEvent event, **/NET** version, on:

C#

```
private void OleEvent(object sender,exontrol.EXRIBBONLib.Item
Itm,exontrol.EXRIBBONLib.OleEvent Ev)
{
}
```

VB

```
Private Sub OleEvent(ByVal sender As System.Object,ByVal Itm As
exontrol.EXRIBBONLib.Item,ByVal Ev As exontrol.EXRIBBONLib.OleEvent) Handles
```



```
OleEvent  
End Sub
```

Syntax for OleEvent event, **/COM** version, on:

```
C# private void OleEvent(object sender,  
AxEXRIBBONLib._IRibbonEvents_OleEventEvent e)  
{  
}
```

```
C++ void OnOleEvent(LPDISPATCH Itm,LPDISPATCH Ev)  
{  
}
```

```
C++ Builder void __fastcall OleEvent(TObject *Sender,Exribbonlib_tlb::Item  
*Itm,Exribbonlib_tlb::IOleEvent *Ev)  
{  
}
```

```
Delphi procedure OleEvent(ASender: TObject; Itm : Item;Ev : IOleEvent);  
begin  
end;
```

```
Delphi 8 (.NET only) procedure OleEvent(sender: System.Object; e:  
AxEXRIBBONLib._IRibbonEvents_OleEventEvent);  
begin  
end;
```

```
Powe... begin event OleEvent(oleobject Itm,oleobject Ev)  
end event OleEvent
```

```
VB.NET Private Sub OleEvent(ByVal sender As System.Object, ByVal e As  
AxEXRIBBONLib._IRibbonEvents_OleEventEvent) Handles OleEvent  
End Sub
```

```
VB6 Private Sub OleEvent(ByVal Itm As EXRIBBONLibCtl.Item,ByVal Ev As  
EXRIBBONLibCtl.IOleEvent)  
End Sub
```

VBA Private Sub OleEvent(ByVal Itm As Object,ByVal Ev As Object)
End Sub

VFP LPARAMETERS Itm,Ev

Xbas... PROCEDURE OnOleEvent(oRibbon,Itm,Ev)
RETURN

Syntax for OleEvent event, **/COM** version (others), on:

Java... <SCRIPT EVENT="OleEvent(Itm,Ev)" LANGUAGE="JScript">
</SCRIPT>

VBSc... <SCRIPT LANGUAGE="VBScript">
Function OleEvent(Itm,Ev)
End Function
</SCRIPT>

Visual
Data... Procedure OnComOleEvent Variant IIItm Variant IIEv
Forward Send OnComOleEvent IIItm IIEv
End_Procedure

Visual
Objects METHOD OCX_OleEvent(Itm,Ev) CLASS MainDialog
RETURN NIL

X++ void onEvent_OleEvent(COM _Itm,COM _Ev)
{
}

XBasic function OleEvent as v (Itm as OLE::Exontrol.Ribbon.1::IItem,Ev as
OLE::Exontrol.Ribbon.1::IOleEvent)
end function

dBASE function nativeObject_OleEvent(Itm,Ev)
return

The following samples shows how to load an ActiveX control ([Exontrol.Calendar](#))

VBA (MS Access, Excell...)

```
With Ribbon1
  With .Items.Add("Calendar",3).SubControl
    .Width = 256
    .Height = 256
    .ControlID = "Exontrol.Calendar"
    .Create
  End With
  .Refresh
End With
```

VB6

```
With Ribbon1
  With .Items.Add("Calendar",3).SubControl
    .Width = 256
    .Height = 256
    .ControlID = "Exontrol.Calendar"
    .Create
  End With
  .Refresh
End With
```

VB.NET

```
With Exribbon1
  With .Items.Add("Calendar",3).SubControl
    .Width = 256
    .Height = 256
    .ControlID = "Exontrol.Calendar"
    .Create()
  End With
  .Refresh()
End With
```

VB.NET for /COM

With AxRibbon1

With .Items.Add("Calendar",3).SubControl

.Width = 256

.Height = 256

.ControlID = "Exontrol.Calendar"

.Create()

End With

.Refresh()

End With

C++

/*

Copy and paste the following directives to your header file as
it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control
Library'

#import <ExRibbon.dll>

using namespace EXRIBBONLib;

*/

EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-

>GetControlUnknown();

EXRIBBONLib::IControlPtr var_Control = spRibbon1->GetItems()-

>Add(L"Calendar",long(3),vtMissing)->GetSubControl();

var_Control->PutWidth(256);

var_Control->PutHeight(256);

var_Control->PutControlID(L"Exontrol.Calendar");

var_Control->Create();

spRibbon1->Refresh();

C++ Builder

Exribbonlib_tlb::IControlPtr var_Control = Ribbon1->Items-

>Add(L"Calendar",TVariant(3),TNoParam())->SubControl;

var_Control->Width = 256;

var_Control->Height = 256;

var_Control->ControlID = L"Exontrol.Calendar";

```
var_Control->Create();  
Ribbon1->Refresh();
```

C#

```
exontrol.EXRIBBONLib.Control var_Control =  
exribbon1.Items.Add("Calendar",3,null).SubControl;  
var_Control.Width = 256;  
var_Control.Height = 256;  
var_Control.ControlID = "Exontrol.Calendar";  
var_Control.Create();  
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>  
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
var var_Control = Ribbon1.Items.Add("Calendar",3,null).SubControl;  
var_Control.Width = 256;  
var_Control.Height = 256;  
var_Control.ControlID = "Exontrol.Calendar";  
var_Control.Create();  
Ribbon1.Refresh();  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload='Init()'>
```

```
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"  
id="Ribbon1"></OBJECT>
```

```
<SCRIPT LANGUAGE="VBScript">
```

```
Function Init()
```

```
    With Ribbon1
```

```
        With .Items.Add("Calendar",3).SubControl
```

```
            .Width = 256
```

```
            .Height = 256
```

```
            .ControlID = "Exontrol.Calendar"
```

```
            .Create
```

```
        End With
```

```
        .Refresh
```

```
    End With
```

```
End Function
```

```
</SCRIPT>
```

```
</BODY>
```

C# for /COM

```
EXRIBBONLib.Control var_Control =  
axRibbon1.Items.Add("Calendar",3,null).SubControl;  
    var_Control.Width = 256;  
    var_Control.Height = 256;  
    var_Control.ControlID = "Exontrol.Calendar";  
    var_Control.Create();  
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()  
{  
    COM com_Control,com_Item;  
    anytype var_Control,var_Item;  
    ;
```

```

super();

var_Item =
COM::createFromObject(exribbon1.Items()).Add("Calendar",COMVariant::createFromInt
com_Item = var_Item;
var_Control = com_Item.SubControl(); com_Control = var_Control;
com_Control.Width(256);
com_Control.Height(256);
com_Control.ControlID("Exontrol.Calendar");
com_Control.Create();
exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

with AxRibbon1 do
begin
  with Items.Add('Calendar',TObject(3),Nil).SubControl do
  begin
    Width := 256;
    Height := 256;
    ControlID := 'Exontrol.Calendar';
    Create();
  end;
  Refresh();
end

```

Delphi (standard)

```

with Ribbon1 do
begin
  with Items.Add('Calendar',OleVariant(3),Null).SubControl do
  begin
    Width := 256;
    Height := 256;
    ControlID := 'Exontrol.Calendar';
    Create();
  end;
end;

```

```
Refresh();  
end
```

VFP

```
with thisform.Ribbon1  
  with .Items.Add("Calendar",3).SubControl  
    .Width = 256  
    .Height = 256  
    .ControlID = "Exontrol.Calendar"  
    .Create  
  endwith  
  .Refresh  
endwith
```

dBASE Plus

```
local oRibbon,var_Control  
  
oRibbon = form.Activex1.nativeObject  
var_Control = oRibbon.Items.Add("Calendar",3).SubControl  
  var_Control.Width = 256  
  var_Control.Height = 256  
  var_Control.ControlID = "Exontrol.Calendar"  
  var_Control.Create()  
oRibbon.Refresh()
```

XBasic (Alpha Five)

```
Dim oRibbon as P  
Dim var_Control as P  
  
oRibbon = topparent:CONTROL_ACTIVEX1.activex  
var_Control = oRibbon.Items.Add("Calendar",3).SubControl  
  var_Control.Width = 256  
  var_Control.Height = 256  
  var_Control.ControlID = "Exontrol.Calendar"
```



```
var_Control.Create()  
oRibbon.Refresh()
```

Visual Objects

```
local var_Control as IControl  
  
var_Control := oDCOCX_Exontrol1:Items:Add("Calendar",3,nil):SubControl  
var_Control.Width := 256  
var_Control.Height := 256  
var_Control.ControlID := "Exontrol.Calendar"  
var_Control.Create()  
oDCOCX_Exontrol1.Refresh()
```

PowerBuilder

```
OleObject oRibbon,var_Control  
  
oRibbon = ole_1.Object  
var_Control = oRibbon.Items.Add("Calendar",3).SubControl  
var_Control.Width = 256  
var_Control.Height = 256  
var_Control.ControlID = "Exontrol.Calendar"  
var_Control.Create()  
oRibbon.Refresh()
```

Visual DataFlex

```
Procedure OnCreate  
Forward Send OnCreate  
Variant voltems  
Get ComItems to voltems  
Handle holtems  
Get Create (RefClass(cComItems)) to holtems  
Set pvComObject of holtems to voltems
```

```

Variant voltem
Get ComAdd of holtems "Calendar" 3 Nothing to voltem
Handle holtem
Get Create (RefClass(cComItem)) to holtem
Set pvComObject of holtem to voltem
    Variant voControl
    Get ComSubControl of holtem to voControl
    Handle hoControl
    Get Create (RefClass(cComControl)) to hoControl
    Set pvComObject of hoControl to voControl
        Set ComWidth of hoControl to 256
        Set ComHeight of hoControl to 256
        Set ComControlID of hoControl to "Exontrol.Calendar"
        Send ComCreate of hoControl
    Send Destroy to hoControl
Send Destroy to holtem
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oControl
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( „{100,100}, {640,480}„, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )

```

```
oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-  
CFBE431702E2}*/
```

```
oRibbon:create(,, {10,60},{610,370} )
```

```
oControl := oRibbon:Items():Add("Calendar",3):SubControl()
```

```
oControl:Width := 256
```

```
oControl:Height := 256
```

```
oControl:ControlID := "Exontrol.Calendar"
```

```
oControl:Create()
```

```
oRibbon:Refresh()
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

event SelectItem (Itm as Item)

Occurs when the user selects the item.

Type	Description
Itm as Item	An Item object being clicked.

The trial/evaluation version of the control limits firing this event. In other words, using the trial/evaluation version won't fire the event every time.

The SelectItem event notifies your application once the user clicks/selects an item. The [ID](#) property specifies the item's identifier. In C++ or VFP, you can use the [Notifier](#) to get notified through the WM_COMMAND message.

Syntax for SelectItem event, **/NET** version, on:

```
C# private void SelectItem(object sender,exontrol.EXRIBBONLib.Item Itm)
{
}
```

```
VB Private Sub SelectItem(ByVal sender As System.Object,ByVal Itm As
exontrol.EXRIBBONLib.Item) Handles SelectItem
End Sub
```

Syntax for SelectItem event, **/COM** version, on:

```
C# private void SelectItem(object sender,
AxEXRIBBONLib._IRibbonEvents_SelectItemEvent e)
{
}
```

```
C++ void OnSelectItem(LPDISPATCH Itm)
{
}
```

```
C++ Builder void __fastcall SelectItem(TObject *Sender,Exribbonlib_tlb::IItem *Itm)
{
}
```

```
Delphi procedure SelectItem(ASender: TObject; Itm : IItem);
begin
```

```
end;
```

Delphi 8
(.NET
only)

```
procedure SelectItem(sender: System.Object; e:  
AxEXRIBBONLib_IRibbonEvents_SelectItemEvent);  
begin  
end;
```

Powe...

```
begin event SelectItem(oleobject Itm)  
end event SelectItem
```

VB.NET

```
Private Sub SelectItem(ByVal sender As System.Object, ByVal e As  
AxEXRIBBONLib_IRibbonEvents_SelectItemEvent) Handles SelectItem  
End Sub
```

VB6

```
Private Sub SelectItem(ByVal Itm As EXRIBBONLibCtl.IItem)  
End Sub
```

VBA

```
Private Sub SelectItem(ByVal Itm As Object)  
End Sub
```

VFP

```
LPARAMETERS Itm
```

Xbas...

```
PROCEDURE OnSelectItem(oRibbon,Itm)  
RETURN
```

Syntax for SelectItem event, **/COM** version (others), on:

Java...

```
<SCRIPT EVENT="SelectItem(Itm)" LANGUAGE="JScript">  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function SelectItem(Itm)  
End Function  
</SCRIPT>
```

Visual
Data...

```
Procedure OnComSelectItem Variant lItm  
Forward Send OnComSelectItem lItm
```

End_Procedure

Visual
Objects

METHOD OCX_SelectItem(Itm) CLASS MainDialog
RETURN NIL

X++

```
void onEvent_SelectItem(COM _Itm)
{
}
```

XBasic

```
function SelectItem as v (Itm as OLE::Exontrol.Ribbon.1::Item)
end function
```

dBASE

```
function nativeObject_SelectItem(Itm)
return
```

The following samples show how you can get notified once the user clicks the item:

VBA (MS Access, Excell...)

' SelectItem event - Occurs when the user selects the item.

```
Private Sub Ribbon1_SelectItem(ByVal Itm As Object)
    With Ribbon1
        Debug.Print( "SelectItem event on Itm object" )
    End With
End Sub
```

```
With Ribbon1
    With .Items
        .Add "Item"
        .Add "Item"
    End With
    .Refresh
End With
```

VB6

' SelectItem event - Occurs when the user selects the item.

```
Private Sub Ribbon1_SelectItem(ByVal Itm As EXRIBBONLibCtl.Item)
```

```

With Ribbon1
    Debug.Print( "SelectedItem event on Itm object" )
End With
End Sub

```

```

With Ribbon1
    With .Items
        .Add "Item"
        .Add "Item"
    End With
    .Refresh
End With

```

VB.NET

' SelectItem event - Occurs when the user selects the item.

Private Sub Exribbon1_SelectItem(ByVal sender As System.Object, ByVal Itm As exontrol.EXRIBBONLib.Item) Handles Exribbon1.**SelectItem**

```

    With Exribbon1
        Debug.Print( "SelectedItem event on Itm object" )
    End With
End Sub

```

```

With Exribbon1
    With .Items
        .Add("Item")
        .Add("Item")
    End With
    .Refresh()
End With

```

VB.NET for /COM

' SelectItem event - Occurs when the user selects the item.

Private Sub AxRibbon1_SelectItem(ByVal sender As System.Object, ByVal e As AxEXRIBBONLib._IRibbonEvents_SelectItemEvent) Handles AxRibbon1.**SelectItem**

```

    With AxRibbon1
        Debug.Print( "SelectedItem event on Itm object" )
    End With

```

```
End With
End Sub
```

```
With AxRibbon1
    With .Items
        .Add("Item")
        .Add("Item")
    End With
    .Refresh()
End With
```

C++

// SelectItem event - Occurs when the user selects the item.

```
void OnSelectItemRibbon1(LPDISPATCH Itm)
{
    /*
        Copy and paste the following directives to your header file as
        it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control
Library'
        #import <ExRibbon.dll>
        using namespace EXRIBBONLib;
    */
    EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
    OutputDebugStringW( L"SelectItem event on Itm object" );
}

EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)-
>GetControlUnknown();
EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();
var_Items->Add(L"Item",vtMissing,vtMissing);
var_Items->Add(L"Item",vtMissing,vtMissing);
spRibbon1->Refresh();
```

C++ Builder

// SelectItem event - Occurs when the user selects the item.

```
void __fastcall TForm1::Ribbon1SelectItem(TObject *Sender,Exribbonlib_tlb::Item
*Itm)
{
    OutputDebugString( L"SelectItem event on Itm object" );
}

Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;
var_Items->Add(L"Item",TNoParam(),TNoParam());
var_Items->Add(L"Item",TNoParam(),TNoParam());
Ribbon1->Refresh();
```

C#

// SelectItem event - Occurs when the user selects the item.

```
private void exribbon1_SelectItem(object sender,exontrol.EXRIBBONLib.Item Itm)
{
    System.Diagnostics.Debug.Print( "SelectItem event on Itm object" );
}

//this.exribbon1.SelectItem += new
exontrol.EXRIBBONLib.exg2antt.SelectItemEventHandler(this.exribbon1_SelectI

exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;
var_Items.Add("Item",null,null);
var_Items.Add("Item",null,null);
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>
<SCRIPT FOR="Ribbon1" EVENT="SelectItem(Itm)" LANGUAGE="JScript">
    alert( "SelectItem event on Itm object" );
</SCRIPT>

<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
```

```
id="Ribbon1"> </OBJECT>
```

```
<SCRIPT LANGUAGE="JScript">
```

```
function Init()
```

```
{
```

```
    var var_Items = Ribbon1.Items;
```

```
    var_Items.Add("Item",null,null);
```

```
    var_Items.Add("Item",null,null);
```

```
    Ribbon1.Refresh();
```

```
}
```

```
</SCRIPT>
```

```
</BODY>
```

VBScript

```
<BODY onload='Init()'>
```

```
<SCRIPT LANGUAGE="VBScript">
```

```
Function Ribbon1_SelectItem(ltm)
```

```
    With Ribbon1
```

```
        alert( "SelectItem event on ltm object" )
```

```
    End With
```

```
End Function
```

```
</SCRIPT>
```

```
<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
```

```
id="Ribbon1"> </OBJECT>
```

```
<SCRIPT LANGUAGE="VBScript">
```

```
Function Init()
```

```
    With Ribbon1
```

```
        With .Items
```

```
            .Add "Item"
```

```
            .Add "Item"
```

```
        End With
```

```
        .Refresh
```

```
    End With
```

```
End Function
```

```
</SCRIPT>
```

```
</BODY>
```

C# for /COM

```
// SelectItem event - Occurs when the user selects the item.
```

```
private void axRibbon1_SelectItem(object sender,  
AxEXRIBBONLib._IRibbonEvents_SelectItemEvent e)
```

```
{  
    System.Diagnostics.Debug.Print( "SelectItem event on Itm object" );  
}
```

```
//this.axRibbon1.SelectItem += new
```

```
AxEXRIBBONLib._IRibbonEvents_SelectItemEventHandler(this.axRibbon1_Select
```

```
EXRIBBONLib.Items var_Items = axRibbon1.Items;
```

```
var_Items.Add("Item",null,null);
```

```
var_Items.Add("Item",null,null);
```

```
axRibbon1.Refresh();
```

X++ (Dynamics Ax 2009)

```
// SelectItem event - Occurs when the user selects the item.
```

```
void onEvent_SelectItem(COM _Itm)
```

```
{  
    ;  
    print( "SelectItem event on Itm object" );  
}
```

```
public void init()
```

```
{  
    COM com_Items;  
    anytype var_Items;  
    ;  
}
```

```

super();

var_Items = exribbon1.Items(); com_Items = var_Items;
    com_Items.Add("Item");
    com_Items.Add("Item");
exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

// SelectItem event - Occurs when the user selects the item.

```

procedure TForm1.AxRibbon1_SelectItem(sender: System.Object; e:
AxEXRIBBONLib._IRibbonEvents_SelectItemEvent);
begin
    with AxRibbon1 do
    begin
        OutputDebugString( 'SelectItem event on Itm object' );
    end
end;

with AxRibbon1 do
begin
    with Items do
    begin
        Add('Item',Nil,Nil);
        Add('Item',Nil,Nil);
    end;
    Refresh();
end

```

Delphi (standard)

// SelectItem event - Occurs when the user selects the item.

```

procedure TForm1.Ribbon1SelectItem(ASender: TObject; Itm : IItem);
begin
    with Ribbon1 do
    begin
        OutputDebugString( 'SelectItem event on Itm object' );
    end
end

```

```

    end
end;

with Ribbon1 do
begin
    with Items do
    begin
        Add('Item',Null,Null);
        Add('Item',Null,Null);
    end;
    Refresh();
end

```

VFP

*** **SelectItem** event - Occurs when the user selects the item. ***

```

LPARAMETERS Itm
with thisform.Ribbon1
    DEBUGOUT( "SelectItem event on Itm object" )
endwith

with thisform.Ribbon1
    with .Items
        .Add("Item")
        .Add("Item")
    endwith
    .Refresh
endwith

```

dBASE Plus

```

/*
with (this.ACTIVEX1.nativeObject)
    SelectItem = class::nativeObject_SelectItem
endwith
*/
// Occurs when the user selects the item.
function nativeObject_SelectItem(Itm)

```

```
local oRibbon
oRibbon = form.Activex1.nativeObject
? "SelectItem event on Itm object"
return
```

```
local oRibbon,var_Items
```

```
oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
var_Items.Add("Item")
var_Items.Add("Item")
oRibbon.Refresh()
```

XBasic (Alpha Five)

' **Occurs when the user selects the item.**

```
function SelectItem as v (Itm as OLE::Exontrol.Ribbon.1::Item)
    Dim oRibbon as P
    oRibbon = topparent:CONTROL_ACTIVEX1.activex
    ? "SelectItem event on Itm object"
end function
```

```
Dim oRibbon as P
Dim var_Items as P
```

```
oRibbon = topparent:CONTROL_ACTIVEX1.activex
var_Items = oRibbon.Items
var_Items.Add("Item")
var_Items.Add("Item")
oRibbon.Refresh()
```

Visual Objects

```
METHOD OCX_Exontrol1SelectItem(Itm) CLASS MainDialog
// SelectItem event - Occurs when the user selects the item.
OutputDebugString(String2Psz( "SelectItem event on Itm object" ))
```

```
RETURN NIL
```

```
local var_Items as Items
```

```
var_Items := oDCOCX_Exontrol1:Items
```

```
var_Items.Add("Item",nil,nil)
```

```
var_Items.Add("Item",nil,nil)
```

```
oDCOCX_Exontrol1:Refresh()
```

PowerBuilder

```
/*begin event SelectItem(oleobject Itm) - Occurs when the user selects the item.*/
```

```
/*
```

```
    OleObject oRibbon
```

```
    oRibbon = ole_1.Object
```

```
    MessageBox("Information",string( "SelectItem event on Itm object" ))
```

```
*/
```

```
/*end event SelectItem*/
```

```
OleObject oRibbon,var_Items
```

```
oRibbon = ole_1.Object
```

```
var_Items = oRibbon.Items
```

```
var_Items.Add("Item")
```

```
var_Items.Add("Item")
```

```
oRibbon.Refresh()
```

Visual DataFlex

```
// Occurs when the user selects the item.
```

```
Procedure OnComSelectItem Variant llltm
```

```
    Forward Send OnComSelectItem llltm
```

```
    Showln "SelectItem event on Itm object"
```

```
End_Procedure
```

```
Procedure OnCreate
```

```

Forward Send OnCreate
Variant voltems
Get ComItems to voltems
Handle holtems
Get Create (RefClass(cComItems)) to holtems
Set pvComObject of holtems to voltems
    Get ComAdd of holtems "Item" Nothing Nothing to Nothing
    Get ComAdd of holtems "Item" Nothing Nothing to Nothing
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

PROCEDURE OnSelectItem(oRibbon,Itm)
    DevOut( "SelectItem event on Itm object" )
RETURN

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltems
    LOCAL oRibbon

    oForm := XbpDialog():new( AppDesktop() )
    oForm:drawingArea:clipChildren := .T.
    oForm:create( „{100,100}, {640,480}„, .F. )
    oForm:close := {|| PostAppEvent( xbeP_Quit )}

    oRibbon := XbpActiveXControl():new( oForm:drawingArea )
    oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
    oRibbon:create(„ {10,60},{610,370} )

```


event UncheckItem (Itm as Item)

Occurs when the user unchecks the item.

Type	Description
Itm as Item	An Item object that indicates the item being un-checked.

The trial/evaluation version of the control limits firing this event. In other words, using the trial/evaluation version won't fire the event every time.

The UncheckItem event notifies your application once the user clicks the item's check-box. The [Check](#) property indicates whether the Item has associated a check box. The [Checked](#) property specifies whether the item is checked or unchecked. Use the [Radio](#) property to display a radio-button on the item. In C++ or VFP, you can use the [Notifier](#) to get notified through the WM_COMMAND message. The [Checked](#) property specifies whether the item is checked or unchecked. The [CheckedItem](#) event notifies once the user checks the item.

Syntax for UncheckItem event, **/NET** version, on:

```
C# private void UncheckItem(object sender,exontrol.EXRIBBONLib.Item Itm)
{
}
```

```
VB Private Sub UncheckItem(ByVal sender As System.Object,ByVal Itm As
exontrol.EXRIBBONLib.Item) Handles UncheckItem
End Sub
```

Syntax for UncheckItem event, **/COM** version, on:

```
C# private void UncheckItem(object sender,
AxEXRIBBONLib._IRibbonEvents_UncheckItemEvent e)
{
}
```

```
C++ void OnUncheckItem(LPDISPATCH Itm)
{
}
```

```
C++ Builder void __fastcall UncheckItem(TObject *Sender,Exribbonlib_tlb::IItem *Itm)
{
}
```

Delphi
procedure UncheckItem(ASender: TObject; Itm : IItem);
begin
end;

Delphi 8
(.NET
only)
procedure UncheckItem(sender: System.Object; e:
AxEXRIBBONLib._IRibbonEvents_UncheckItemEvent);
begin
end;

Powe...
begin event UncheckItem(oleobject Itm)
end event UncheckItem

VB.NET
Private Sub UncheckItem(ByVal sender As System.Object, ByVal e As
AxEXRIBBONLib._IRibbonEvents_UncheckItemEvent) Handles UncheckItem
End Sub

VB6
Private Sub UncheckItem(ByVal Itm As EXRIBBONLibCtl.IItem)
End Sub

VBA
Private Sub UncheckItem(ByVal Itm As Object)
End Sub

VFP
LPARAMETERS Itm

Xbas...
PROCEDURE OnUncheckItem(oRibbon,Itm)
RETURN

Syntax for UncheckItem event, **/COM** version (others), on:

Java...
<SCRIPT EVENT="UncheckItem(Itm)" LANGUAGE="JScript">
</SCRIPT>

VBSc...
<SCRIPT LANGUAGE="VBScript">
Function UncheckItem(Itm)
End Function
</SCRIPT>

Visual
Data...

```
Procedure OnComUncheckItem Variant lItm  
    Forward Send OnComUncheckItem lItm  
End_Procedure
```

Visual
Objects

```
METHOD OCX_UncheckItem(ltm) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_UncheckItem(COM _ltm)  
{  
}
```

XBasic

```
function UncheckItem as v (ltm as OLE::Exontrol.Ribbon.1::IItem)  
end function
```

dBASE

```
function nativeObject_UncheckItem(ltm)  
return
```

The following samples show how you can get notified once the user un-checks an item:

VBA (MS Access, Excell...)

' UncheckItem event - Occurs when the user unchecks the item.

```
Private Sub Ribbon1_UncheckItem(ByVal ltm As Object)  
    With Ribbon1  
        Debug.Print( "UncheckItem event on ltm object" )  
    End With  
End Sub
```

```
With Ribbon1  
    With .Items  
        .Add("Item").Check = True  
        .Add("Item").Check = True  
    End With  
    .Refresh  
End With
```

VB6

' **UncheckItem event - Occurs when the user unchecks the item.**

```
Private Sub Ribbon1_UncheckItem(ByVal Itm As EXRIBBONLibCtl.Item)
```

```
    With Ribbon1
```

```
        Debug.Print( "UncheckItem event on Itm object" )
```

```
    End With
```

```
End Sub
```

```
With Ribbon1
```

```
    With .Items
```

```
        .Add("Item").Check = True
```

```
        .Add("Item").Check = True
```

```
    End With
```

```
    .Refresh
```

```
End With
```

VB.NET

' **UncheckItem event - Occurs when the user unchecks the item.**

```
Private Sub Exribbon1_UncheckItem(ByVal sender As System.Object,ByVal Itm As  
exontrol.EXRIBBONLib.Item) Handles Exribbon1.UncheckItem
```

```
    With Exribbon1
```

```
        Debug.Print( "UncheckItem event on Itm object" )
```

```
    End With
```

```
End Sub
```

```
With Exribbon1
```

```
    With .Items
```

```
        .Add("Item").Check = True
```

```
        .Add("Item").Check = True
```

```
    End With
```

```
    .Refresh()
```

```
End With
```

VB.NET for /COM

' **UncheckItem event - Occurs when the user unchecks the item.**

Private Sub AxRibbon1_UncheckItem(ByVal sender As System.Object, ByVal e As AxEXRIBBONLib._IRibbonEvents_UncheckItemEvent) Handles

AxRibbon1.**UncheckItem**

With AxRibbon1

Debug.Print("UncheckItem event on itm object")

End With

End Sub

With AxRibbon1

With .Items

.Add("Item").Check = True

.Add("Item").Check = True

End With

.Refresh()

End With

C++

// UncheckItem event - Occurs when the user unchecks the item.

void OnUncheckItemRibbon1(LPDISPATCH Itm)

{

/*

Copy and paste the following directives to your header file as

it defines the namespace 'EXRIBBONLib' for the library: 'ExRibbon 1.0 Control

Library'

#import <ExRibbon.dll>

using namespace EXRIBBONLib;

*/

EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)->GetControlUnknown();

OutputDebugStringW(L"UncheckItem event on itm object");

}

EXRIBBONLib::IRibbonPtr spRibbon1 = GetDlgItem(IDC_RIBBON1)->GetControlUnknown();

EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();

EXRIBBONLib::IItemsPtr var_Items = spRibbon1->GetItems();

var_Items->Add(L"Item",vtMissing,vtMissing)->PutCheck(VARIANT_TRUE);

```
var_Items->Add(L"Item",vtMissing,vtMissing)->PutCheck(VARIANT_TRUE);  
spRibbon1->Refresh();
```

C++ Builder

```
// UncheckItem event - Occurs when the user unchecks the item.  
void __fastcall TForm1::Ribbon1UncheckItem(TObject *Sender,Exribbonlib_tlb::Item  
*Itm)  
{  
    OutputDebugString( L"UncheckItem event on Itm object" );  
}  
  
Exribbonlib_tlb::IItemsPtr var_Items = Ribbon1->Items;  
var_Items->Add(L"Item",TNoParam(),TNoParam())->Check = true;  
var_Items->Add(L"Item",TNoParam(),TNoParam())->Check = true;  
Ribbon1->Refresh();
```

C#

```
// UncheckItem event - Occurs when the user unchecks the item.  
private void exribbon1_UncheckItem(object sender,exontrol.EXRIBBONLib.Item Itm)  
{  
    System.Diagnostics.Debug.Print( "UncheckItem event on Itm object" );  
}  
  
//this.exribbon1.UncheckItem += new  
exontrol.EXRIBBONLib.exg2antt.UncheckItemEventHandler(this.exribbon1_Uncl  
  
  
exontrol.EXRIBBONLib.Items var_Items = exribbon1.Items;  
var_Items.Add("Item",null,null).Check = true;  
var_Items.Add("Item",null,null).Check = true;  
exribbon1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>
<SCRIPT FOR="Ribbon1" EVENT="UncheckItem(Itm)" LANGUAGE="JScript">
    alert( "UncheckItem event on Itm object" );
</SCRIPT>

<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    var var_Items = Ribbon1.Items;
    var_Items.Add("Item",null,null).Check = true;
    var_Items.Add("Item",null,null).Check = true;
    Ribbon1.Refresh();
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload='Init()'>
<SCRIPT LANGUAGE="VBScript">
Function Ribbon1_UncheckItem(Itm)
    With Ribbon1
        alert( "UncheckItem event on Itm object" )
    End With
End Function
</SCRIPT>

<OBJECT CLASSID="clsid:DDF58CFA-750F-45E0-8A00-CFBE431702E2"
id="Ribbon1"> </OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Ribbon1
```



```

With .Items
    .Add("Item").Check = True
    .Add("Item").Check = True
End With
.Refresh
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

// UncheckItem event - Occurs when the user unchecks the item.
private void axRibbon1_UncheckItem(object sender,
AxEXRIBBONLib._IRibbonEvents_UncheckItemEvent e)
{
    System.Diagnostics.Debug.Print( "UncheckItem event on Itm object" );
}

//this.axRibbon1.UncheckItem += new
AxEXRIBBONLib._IRibbonEvents_UncheckItemEventHandler(this.axRibbon1_Un

EXRIBBONLib.Items var_Items = axRibbon1.Items;
    var_Items.Add("Item",null,null).Check = true;
    var_Items.Add("Item",null,null).Check = true;
axRibbon1.Refresh();

```

X++ (Dynamics Ax 2009)

```

// UncheckItem event - Occurs when the user unchecks the item.
void onEvent_UncheckItem(COM _Itm)
{
    ;
    print( "UncheckItem event on Itm object" );
}

```

```

public void init()
{
    COM com_Item,com_Items;
    anytype var_Item,var_Items;
    ;

    super();

    var_Items = exribbon1.Items(); com_Items = var_Items;
    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
    com_Item.Check(true);
    var_Item = COM::createFromObject(com_Items.Add("Item")); com_Item =
var_Item;
    com_Item.Check(true);
    exribbon1.Refresh();
}

```

Delphi 8 (.NET only)

```

// UncheckItem event - Occurs when the user unchecks the item.
procedure TForm1.AxRibbon1_UncheckItem(sender: System.Object; e:
AxEXRIBBONLib._IRibbonEvents_UncheckItemEvent);
begin
    with AxRibbon1 do
    begin
        OutputDebugString( 'UncheckItem event on itm object' );
    end
end;

with AxRibbon1 do
begin
    with Items do
    begin
        Add('Item',Nil,Nil).Check := True;
        Add('Item',Nil,Nil).Check := True;
    end;
end;

```

```
Refresh();  
end
```

Delphi (standard)

// UncheckItem event - Occurs when the user unchecks the item.

```
procedure TForm1.Ribbon1UncheckItem(ASender: TObject; Itm : IItem);  
begin  
    with Ribbon1 do  
    begin  
        OutputDebugString( 'UncheckItem event on Itm object' );  
    end  
end;  
  
with Ribbon1 do  
begin  
    with Items do  
    begin  
        Add('Item',Null,Null).Check := True;  
        Add('Item',Null,Null).Check := True;  
    end;  
    Refresh();  
end
```

VFP

*** **UncheckItem** event - Occurs when the user unchecks the item. ***

```
LPARAMETERS Itm  
    with thisform.Ribbon1  
        DEBUGOUT( "UncheckItem event on Itm object" )  
    endwith  
  
with thisform.Ribbon1  
    with .Items  
        .Add("Item").Check = .T.  
        .Add("Item").Check = .T.  
    endwith  
    .Refresh
```

```
endwith
```

dBASE Plus

```
/*
with (this.ACTIVEX1.nativeObject)
    UncheckItem = class::nativeObject_UncheckItem
endwith
*/
// Occurs when the user unchecks the item.
function nativeObject_UncheckItem(ltm)
    local oRibbon
    oRibbon = form.Activex1.nativeObject
    ? "UncheckItem event on ltm object"
return

local oRibbon,var_Item,var_Item1,var_Items

oRibbon = form.Activex1.nativeObject
var_Items = oRibbon.Items
// var_Items.Add("Item").Check = true
var_Item = var_Items.Add("Item")
with (oRibbon)
    TemplateDef = [Dim var_Item]
    TemplateDef = var_Item
    Template = [var_Item.Check = true]
endwith
// var_Items.Add("Item").Check = true
var_Item1 = var_Items.Add("Item")
with (oRibbon)
    TemplateDef = [Dim var_Item1]
    TemplateDef = var_Item1
    Template = [var_Item1.Check = true]
endwith
oRibbon.Refresh()
```

XBasic (Alpha Five)

' Occurs when the user unchecks the item.

```
function UncheckItem as v (Itm as OLE::Exontrol.Ribbon.1::Item)
```

```
    Dim oRibbon as P
```

```
    oRibbon = topparent:CONTROL_ACTIVEX1.activex
```

```
    ? "UncheckItem event on Itm object"
```

```
end function
```

```
Dim oRibbon as P
```

```
Dim var_Item as P
```

```
Dim var_Item1 as P
```

```
Dim var_Items as P
```

```
oRibbon = topparent:CONTROL_ACTIVEX1.activex
```

```
var_Items = oRibbon.Items
```

```
    ' var_Items.Add("Item").Check = .t.
```

```
    var_Item = var_Items.Add("Item")
```

```
    oRibbon.TemplateDef = "Dim var_Item"
```

```
    oRibbon.TemplateDef = var_Item
```

```
    oRibbon.Template = "var_Item.Check = True"
```

```
    ' var_Items.Add("Item").Check = .t.
```

```
    var_Item1 = var_Items.Add("Item")
```

```
    oRibbon.TemplateDef = "Dim var_Item1"
```

```
    oRibbon.TemplateDef = var_Item1
```

```
    oRibbon.Template = "var_Item1.Check = True"
```

```
oRibbon.Refresh()
```

Visual Objects

```
METHOD OCX_Exontrol1UncheckItem(Itm) CLASS MainDialog
```

```
    // UncheckItem event - Occurs when the user unchecks the item.
```

```
    OutputDebugString(String2Psz( "UncheckItem event on Itm object" ))
```

```
RETURN NIL
```

```
local var_Items as IItems
```

```

var_Items := oDCOCX_Exontrol1:Items
  var_Items:Add("Item",nil,nil):Check := true
  var_Items:Add("Item",nil,nil):Check := true
oDCOCX_Exontrol1:Refresh()

```

PowerBuilder

```

/*begin event UncheckItem(oleobject ltm) - Occurs when the user unchecks the
item.*/
/*
  OleObject oRibbon
  oRibbon = ole_1.Object
  MessageBox("Information",string( "UncheckItem event on ltm object" ))
*/
/*end event UncheckItem*/

```

```
OleObject oRibbon,var_Items
```

```

oRibbon = ole_1.Object
var_Items = oRibbon.Items
  var_Items.Add("Item").Check = true
  var_Items.Add("Item").Check = true
oRibbon.Refresh()

```

Visual DataFlex

// Occurs when the user unchecks the item.

```

Procedure OnComUncheckItem Variant lltm
  Forward Send OnComUncheckItem lltm
  Showln "UncheckItem event on ltm object"
End_Procedure

```

```

Procedure OnCreate
  Forward Send OnCreate
  Variant voltems

```

```

Get ComItems to voltems
Handle holtems
Get Create (RefClass(cComItems)) to holtems
Set pvComObject of holtems to voltems
    Variant voltem
    Get ComAdd of holtems "Item" Nothing Nothing to voltem
    Handle holtem
    Get Create (RefClass(cComItem)) to holtem
    Set pvComObject of holtem to voltem
        Set ComCheck of holtem to True
    Send Destroy to holtem
    Variant voltem1
    Get ComAdd of holtems "Item" Nothing Nothing to voltem1
    Handle holtem1
    Get Create (RefClass(cComItem)) to holtem1
    Set pvComObject of holtem1 to voltem1
        Set ComCheck of holtem1 to True
    Send Destroy to holtem1
Send Destroy to holtems
Send ComRefresh
End_Procedure

```

XBase++

```

PROCEDURE OnUncheckItem(oRibbon,Itm)
    DevOut( "UncheckItem event on Itm object" )
RETURN

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
    LOCAL oForm
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
    LOCAL oltems
    LOCAL oRibbon

```

```

oForm := XbpDialog():new( AppDesktop() )
oForm:drawingArea:clipChildren := .T.
oForm:create( ,, {100,100}, {640,480},,, .F. )
oForm:close := {|| PostAppEvent( xbeP_Quit )}

oRibbon := XbpActiveXControl():new( oForm:drawingArea )
oRibbon:CLSID := "Exontrol.Ribbon.1" /*{DDF58CFA-750F-45E0-8A00-
CFBE431702E2}*/
oRibbon:create(,, {10,60},{610,370} )

oRibbon:UncheckItem := {||Itm| OnUncheckItem(oRibbon,Itm)} /*Occurs when
the user unchecks the item.*/

oItems := oRibbon:Items()
oItems:Add("Item"):Check := .T.
oItems:Add("Item"):Check := .T.
oRibbon:Refresh()

oForm:Show()
DO WHILE nEvent != xbeP_Quit
nEvent := AppEvent( @mp1, @mp2, @oXbp )
oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```


Expressions

An expression is a string which defines a formula or criteria, that's evaluated at runtime. The expression may be a combination of variables, constants, strings, dates and operators/functions. For instance `1000 format ``` gets `1,000.00` for US format, while `1.000,00` is displayed for German format.

The Exontrol's [eXPression](#) component is a syntax-editor that helps you to define, view, edit and evaluate expressions. Using the eXPression component you can easily view or check if the expression you have used is syntactically correct, and you can evaluate what is the result you get giving different values to be tested. The Exontrol's eXPression component can be used as an user-editor, to configure your applications.

Usage examples:

- `100 + 200`, adds numbers and returns `300`
- `"100" + 200`, concatenates the strings, and returns `"100200"`
- `currency(1000)` displays the value in currency format based on the current regional setting, such as `"$1,000.00"` for US format.
- `1000 format ``` gets `1,000.00` for English format, while `1.000,00` is displayed for German format
- `1000 format `2|.|3|,`` always gets `1,000.00` no matter of settings in the control panel.
- `upper("string")` converts the giving string in uppercase letters, such as `"STRING"`
- `date(dateS('3/1/' + year(9:=#1/1/2018#)) + ((1:=(((255 - 11 * (year(=:9) mod 19)) - 21) mod 30) + 21) + (=:1 > 48 ? -1 : 0) + 6 - ((year(=:9) + int(year(=:9) / 4)) + =:1 + (=:1 > 48 ? -1 : 0) + 1) mod 7))` returns the date the Easter Sunday will fall, for year 2018. In this case the expression returns `#4/1/2018#`. If `#1/1/2018#` is replaced with `#1/1/2019#`, the expression returns `#4/21/2019#`.

Listed bellow are all predefined constants, operators and functions the general-expression supports:

The constants can be represented as:

- numbers in **decimal** format (where dot character specifies the decimal separator). For instance: `-1`, `100`, `20.45`, `.99` and so on
- numbers in **hexa-decimal** format (preceded by `0x` or `0X` sequence), uses sixteen distinct symbols, most often the symbols 0-9 to represent values zero to nine, and A, B, C, D, E, F (or alternatively a, b, c, d, e, f) to represent values ten to fifteen. Hexadecimal numerals are widely used by computer system designers and programmers. As each hexadecimal digit represents four binary digits (bits), it allows a more human-friendly representation of binary-coded values. For instance, `0xFF`,

0x00FF00, and so so.

- **date-time** in format **#mm/dd/yyyy hh:mm:ss#**, For instance, **#1/31/2001 10:00#** means the **January 31th, 2001, 10:00 AM**
- **string**, if it starts / ends with any of the ' or ` or " characters. If you require the starting character inside the string, it should be escaped (preceded by a \ character). For instance, **`Mihai`**, **"Filimon"**, **'has'**, **"\"a quote\""**, and so on

The predefined constants are:

- **bias** (BIAS constant), defines the difference, in minutes, between Coordinated Universal Time (UTC) and local time. For example, Middle European Time (MET, GMT+01:00) has a time zone bias of "-60" because it is one hour ahead of UTC. Pacific Standard Time (PST, GMT-08:00) has a time zone bias of "+480" because it is eight hours behind UTC. For instance, **date(value - bias/24/60)** converts the UTC time to local time, or **date(date('now') + bias/24/60)** converts the current local time to UTC time. For instance, **"date(value - bias/24/60)"** converts the value date-time from UTC to local time, while **"date(value + bias/24/60)"** converts the local-time to UTC time.
- **dpi** (DPI constant), specifies the current DPI setting. and it indicates the minimum value between **dpix** and **dpiy** constants. For instance, if current DPI setting is 100%, the dpi constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression **value * dpi** returns the value if the DPI setting is 100%, or **value * 1.5** in case, the DPI setting is 150%
- **dpix** (DPIX constant), specifies the current DPI setting on x-scale. For instance, if current DPI setting is 100%, the dpix constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression **value * dpix** returns the value if the DPI setting is 100%, or **value * 1.5** in case, the DPI setting is 150%
- **dpiy** (DPIY constant), specifies the current DPI setting on y-scale. For instance, if current DPI setting is 100%, the dpiy constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression **value * dpiy** returns the value if the DPI setting is 100%, or **value * 1.5** in case, the DPI setting is 150%

The supported binary arithmetic operators are:

- ***** (multiplicity operator), priority 5
- **/** (divide operator), priority 5
- **mod** (remainder operator), priority 5
- **+** (addition operator), priority 4 (concatenates two strings, if one of the operands is of string type)
- **-** (subtraction operator), priority 4

The supported unary boolean operators are:

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

The supported binary boolean operators are:

- **or** (or operator), priority 2
- **and** (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 not found, else the associated element in the collection if it is found. The syntax for array operator is

expression array (c1,c2,c3,...cn)

, where the c1, c2, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the `month(value)-1 array ('J','F','M','A','M','Jun','J','A','S','O','N','D')` is equivalent with `month(value)-1 case (default:"; 0:'J';1:'F';2:'M';3:'A';4:'M';5:'Jun';6:'J';7:'A';8:'S';9:'O';10:'N';11:'D')`.

- **in (include operator)**, specifies whether an element is found in a set of constant elements. The in operator returns -1 (True) if the element is found, else 0 (false) is retrieved. The syntax for in operator is

expression in (c1,c2,c3,...cn)

, where the c1, c2, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the `value in (11,22,33,44,13)` is equivalent with `(expression = 11) or (expression = 22) or (expression = 33) or (expression = 44) or (expression = 13)`. The in operator is not a time consuming as the equivalent or version is, so when you have large number of constant elements it is recommended using the in operator. Shortly, if the collection of elements has 1000 elements the in operator could take up to 8 operations in order to find if an element fits the set, else if the or

statement is used, it could take up to 1000 operations to check, so by far, the *in* operator could save time on finding elements within a collection.

- **switch** (*switch operator*), returns the value being found in the collection, or a predefined value if the element is not found (default). The syntax for *switch* operator is

expression switch (default,c1,c2,c3,...,cn)

, where the c1, c2, ... are constant elements, and the default is a constant element being returned when the element is not found in the collection. The constant elements could be numeric, date or string expressions. The equivalent syntax is "%0 = c 1 ? c 1 : (%0 = c 2 ? c 2 : (... ? . : default))". The *switch* operator is very similar with the *in* operator excepts that the first element in the switch is always returned by the statement if the element is not found, while the returned value is the value itself instead -1. For instance, the *%0 switch ('not found',1,4,7,9,11)* gets 1, 4, 7, 9 or 11, or 'not found' for any other value. As the *in* operator the *switch* operator uses binary searches for fitting the element, so it is quicker than *iif* (immediate if operator) alternative.

- **case()** (*case operator*) returns and executes one of n expressions, depending on the evaluation of the expression (*IIF* - immediate IF operator is a binary *case()* operator). The syntax for *case()* operator is:

expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)

If the default part is missing, the *case()* operator returns the value of the expression if it is not found in the collection of cases (c1, c2, ...). For instance, if the value of expression is not any of c1, c2, the *default_expression* is executed and returned. If the value of the expression is c1, then the *case()* operator executes and returns the *expression1*. The *default, c1, c2, c3, ...* must be constant elements as numbers, dates or strings. For instance, the *date(shortdate(value)) case (default:0 ; #1/1/2002#:1 ; #2/1/2002#:1; #4/1/2002#:1; #5/1/2002#:1)* indicates that only #1/1/2002#, #2/1/2002#, #4/1/2002# and #5/1/2002# dates returns 1, since the others returns 0. For instance the following sample specifies the hour being non-working for specified dates: *date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)* statement indicates the working hours for dates as follows:

- #4/1/2009#, from hours 06:00 AM to 12:00 PM
- #4/5/2009#, from hours 07:00 AM to 10:00 AM and hours 03:00PM, 04:00PM, 06:00PM and 10:00PM
- #5/1/2009#, from hours 12:00 AM to 08:00 AM

The *in*, *switch* and *case()* use binary search to look for elements so they are faster than using *if* and *or* expressions. Obviously, the priority of the operations inside the expression is determined by () parenthesis and the priority for each operator.

The supported conversion unary operators are:

- **type** (unary operator) retrieves the type of the object. The type operator may return any of the following: 0 - empty (not initialized), 1 - null, 2 - short, 3 - long, 4 - float, 5 - double, 6 - currency, **7 - date**, **8 - string**, 9 - object, 10 - error, **11 - boolean**, 12 - variant, 13 - any, 14 - decimal, 16 - char, 17 - byte, 18 - unsigned short, 19 - unsigned long, 20 - long on 64 bits, 21 - unsigned long on 64 bits. For instance `type(%1) = 8` specifies the cells (on the column with the index 1) that contains string values.
- **str** (unary operator) converts the expression to a string. The str operator converts the expression to a string. For instance, the `str(-12.54)` returns the string "-12.54".
- **dbl** (unary operator) converts the expression to a number. The dbl operator converts the expression to a number. For instance, the `dbl("12.54")` returns 12.54
- **date** (unary operator) converts the expression to a date, based on your regional settings. For instance, the `date(``)` gets the current date (no time included), the `date(now)` gets the current date-time, while the `date("01/01/2001")` returns #1/1/2001#
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS. For instance, the `dateS("01/01/2001 14:00:00")` returns #1/1/2001 14:00:00#
- **hex** (unary operator) converts the giving string from hexa-representation to a numeric value, or converts the giving numeric value to hexa-representation as string. For instance, `hex(`FF`)` returns 255, while the `hex(255)` or `hex(0xFF)` returns the `FF` string. The `hex(hex(`FFFFFFFF`))` always returns `FFFFFFFF` string, as the second hex call converts the giving string to a number, and the first hex call converts the returned number to string representation (hexa-representation).

The bitwise operators for numbers are:

- a **bitand** b (binary operator) computes the AND operation on bits of a and b, and returns the unsigned value. For instance, `0x01001000 bitand 0x10111000` returns `0x00001000`.
- a **bitor** b (binary operator) computes the OR operation on bits of a and b, and returns the unsigned value. For instance, `0x01001000 bitor 0x10111000` returns `0x11111000`.
- a **bitxor** b (binary operator) computes the XOR (exclusive-OR) operation on bits of a and b, and returns the unsigned value. For instance, `0x01110010 bitxor 0x10101010` returns `0x11011000`.
- a **bitshift** (b) (binary operator) shifts every bit of a value to the left if b is negative, or to the right if b is positive, for b times, and returns the unsigned value. For instance, `128 bitshift 1` returns 64 (dividing by 2) or `128 bitshift (-1)` returns 256 (multiplying by 2)

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 **rpadd** 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 .