

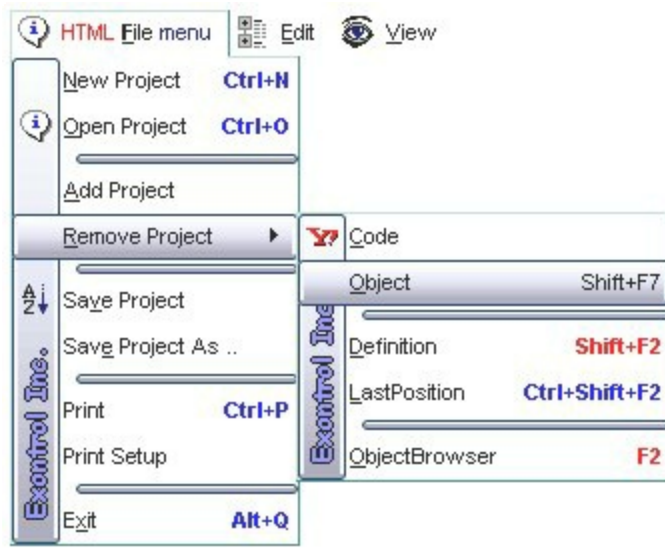


ExMenu

The ExMenu component is a complete new type of control that displays and handles more than a menu for your application. The ExMenu simulates menu bar, popup menu, options button, toolbar, pick list, and more. Make your application more intuitive using the ExMenu component.

Features include:

- **WYSIWYG** editor that helps you to build your menu at design time, quick and easy
- **Skinnable Interface** support (ability to apply a skin to the any background part)
- **Chevron** support
- HTML Multiple Lines ToolTip support
- Ability to display any ActiveX control inside sub menus
- XP shadow border effect.
- Ability to use **built-in HTML format** inside item.
- Partially Translucent support.
- Accelerator keys support.
- Mnemonic (shortcut) keys support.
- Ability to load the list of icons using BASE64 encoded strings.
- Ability to pop up the sub menus.
- Ability to display a picture on menu's background.
- Ability to add an edit control to any menu item
- Ability to display the menu horizontally and vertically as well
- Mouse wheel support
- Ability to insert more than a menu to your form or dialog
- The menu's background and foreground colors are customizable
- Ability to add, remove or change the items, at runtime as well
- Standard, button and flat appearance, like Microsoft .NET menu
- Support for multiple type of borders
- Unlimited color options for any item
- Font attributes(bold, italic, ...) for any menu item
- Support for checkboxes and bullets as well
- Adding to add icons to your menu by drag and drop
- Ability to scroll the menu items



Ž ExMenu 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 item's alignment. Use the [Alignment](#) property to align the caption of the item.

Name	Value	Description
exLeft	0	The item is left aligned.
exCenter	1	The item is centered.
exRight	2	The item is right aligned.

constants AppearanceEnum

The AppearanceEnum defines the control's appearance. Use the [Appearance](#) property to specify the control's appearance.

Name	Value	Description
Normal	0	Normal appearance
Flat	1	Flat appearance, like in the Microsoft NET environment.
Button	2	Button appearance.

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
exSeparatorItem	0	Specifies the visual appearance for a separator item.
exButtonItem	1	Specifies the visual appearance for an item, when the Appearance property is Button.
exSelectRootItem	2	Specifies the visual appearance for a top level item, when the Appearance property is Flat.
exToolTipAppearance	64	Indicates the visual appearance of the borders of the tooltips. Use the ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the ToolTipWidth property to specify the width of the tooltip window. The ToolTipDelay property specifies the time in ms that passes before the ToolTip appears.
exToolTipBackColor	65	Specifies the tooltip's background color.
exToolTipForeColor	66	Specifies the tooltip's foreground color.

constants BorderEnum

The BorderEnum type defines the control's border. Use the NoBorder value to hide the control's border. Use the [Border](#) property to specify the control's border.

Name	Value	Description
NoBorder	0	No border
FlatBorder	1	Flat
SunkenBorder	2	Sunken
RaisedBorder	3	Raised
EtchedBorder	4	Etched
BumpBorder	5	Bump
ShadowBorder	6	Shadow
InsetBorder	7	Inset
SingleBorder	8	Single

constants ChevronEnum

Specifies whether the control displays the chevron or scroll on the menu bar. Use the [AllowChevron](#) property to display chevron or scroll on the menu bar.

Name	Value	Description
exChevron	-1	The control displays the chevron on the menu bar.
exDefault	0	The control displays no chevron or scroll on the menu bar.
exScroll	1	The control displays the scroll on the menu bar.

constants CloseOnEnum

Specifies how a popup menu that hosts an ActiveX control is closed using the mouse. The [CloseOn](#) property indicates how an ActiveX popup menu is closed.

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.
exLButtonDbIClk	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.
exRButtonDbIClk	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.
exMButtonDbIClk	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.
exDbIClick	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 **OpenModeEnum**

Defines the way how the menu bar's sub menus are displayed.

Name	Value	Description
Vertical	0	The popup menus get displayed from top to bottom.
Horizontal	1	The popup menus get displayed from left to right.

constants **OpenOnClickEnum**

The `OpenOnClickEnum` expression specifies how the user opens the menus. The [OpenOnClick](#) property indicates the way the user opens the menus.

Name	Value	Description
<code>exClickMenuBar</code>	-1	The control opens the popup when the user clicks the menu bar.
<code>exHoverMenuBar</code>	0	The control opens the popup when the cursor hovers the menu bar.
<code>exAlwaysClick</code>	1	Any popup requires a click to be opened.

constants **PictureDisplayEnum**

Specifies how the control's picture is arranged. Use the [Picture](#) property to put a picture on the control's background. Use the [Picture](#) property to put a picture on a popup menu's background.

Name	Value	Description
UpperLeft	0	UpperLeft
UpperCenter	1	UpperCenter
UpperRight	2	UpperRight
MiddleLeft	16	MiddleLeft
MiddleCenter	17	MiddleCenter
MiddleRight	18	MiddleRight
LowerLeft	32	LowerLeft
LowerCenter	33	LowerCenter
LowerRight	34	LowerRight
Tile	48	Tile
Stretch	49	Stretch

constants ScrollOnWheelEnum

The ScrollOnWheelEnum type supports the following values:

Name	Value	Description
exScrollOnWheelDisabled	0	The menu is not scrolled when the user rotates the mouse wheel.
exScrollOnWheelEnabled	1	The menu gets scrolled when the user rotates the mouse wheel.
exScrollOnWheelJumpToArrow17		The mouse cursor position is changed once the user scrolls the menu.

constants SelectOnEnum

Specifies the type of selecting items when user presses or release the mouse button. Use the [SelectOn](#) property to specify whether the control selects an item when user presses or releases the mouse button.

Name	Value	Description
exMouseDown	-1	The control selects the item when user presses the mouse button.
exMouseUp	0	The control selects the item when user releases the mouse button.

constants ShowPopupEffectEnum

The ShowPopupEffectEnum value indicates the effect to be shown, when the user clicks an item with a sub-menu associated. The [ScrollOnDrop](#) property indicates the effect to be applied when the popup menu is first shown.

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 ItemTypeEnum

Defines the type of the item being added. The [Add](#) property adds a new item of specified type. Use the [Items](#) property to access the collection of items in the control.

Name	Value	Description
Default	0	Specifies a non popup item.
Separator	1	Specifies a separator item.
SubMenu	2	Specifies a sub menu.
SubControl	3	Specifies a popup menu that hosts an ActiveX control.

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.

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>
Skin as Variant	<p>A string expression that may indicates one of the following:</p> <ol style="list-style-type: none">1. an Windows XP Theme part, it should start with "XP:". 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. In this case the format of the Skin parameter should be: "XP: Control/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 like listed at the end of the document. This option is available only on Windows XP that supports Themes API.2. a copy of another skin with different coordinates, if it begins with "CP:". For instance, you may need to display a specified skin on a smaller rectangle. In this case, the string starts with "CP:", and contains the following "<u>CP:n l t r b</u>", where the n is the identifier being copied, the l, t, r, and b indicate the left, top, right and bottom coordinates being used to adjust the rectangle where the skin is displayed.3. the path to the skin file (*.ebn). The Exontrol's exButton component installs a skin builder that should be used to create new skins4. the BASE64 encoded string that holds a skin file (*.ebn). Use the Exontrol's exImages tool to build BASE 64 encoded strings on the skin file (*.ebn) you have created. Loading the skin from a file (eventually uncompressed file) is always faster then loading from a BASE64 encoded string <p>A byte[] or safe arrays of VT_I1 or VT_UI1 expression that indicates the content of the EBN file. You can use this</p>

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 of 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)).

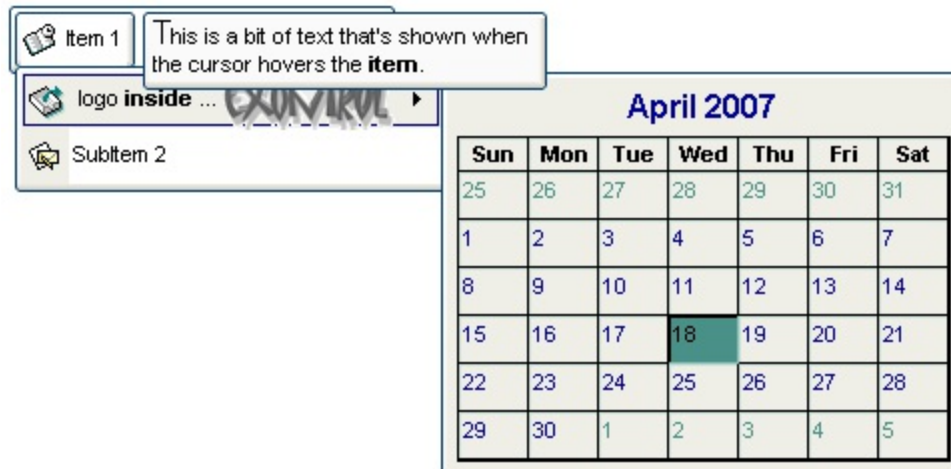
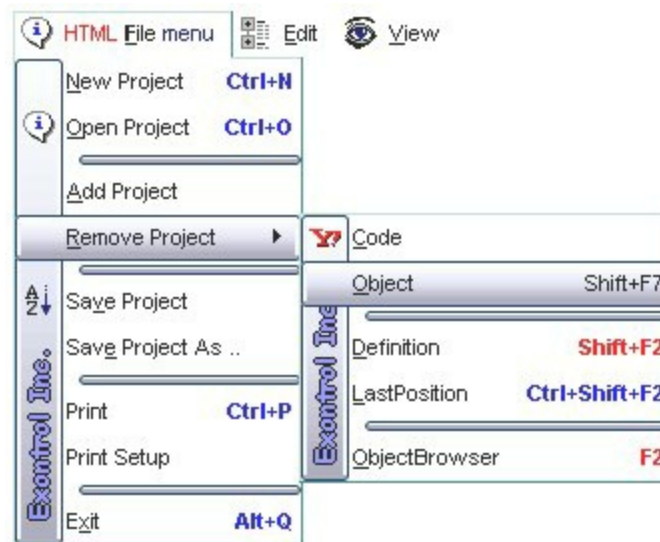
Return

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 its 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.




The identifier you choose for the skin is very important to be used in the background properties like explained below. Shortly, the color properties uses 4 bytes (DWORD, double WORD, and so on) to hold a RGB value. More than that, the first byte (most

significant byte in the color) is used only to specify system color. if the first bit in the byte is 1, the rest of bits indicates the index of the system color being used. So, we use the last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. So, since the 7 bits can cover 127 values, excluding 0, we have 126 possibilities to store an identifier in that byte. This way, a DWORD expression indicates the background color stored in RRGGBB format and the index of the skin (ID parameter) in the last 7 bits in the high significant byte of the color. For instance, the BackColor = BackColor Or &H2000000 indicates that we apply the skin with the index 2 using the old color, to the object that BackColor is applied.

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

- borders, [MenuBarBorder](#) and [Border](#) properties
- selected item, [SelBackColor](#) property
- shadow area, [ShadowColor](#) property
- item, [BackColor](#) property
- separator item, tooltips, top level items, [Background](#) property
- and so on.

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

```
With ExMenu1
  With .VisualAppearance
    .Add &H23, App.Path + "\selected.ebn"
  End With
  .SelForeColor = RGB(0, 0, 0)
  .SelBackColor = &H23000000
End With
```

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

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

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

```
With thisform.ExMenu1
```

```
With .VisualAppearance
```

```
  .Add(35, "D:\Temp\ExMenu_Help\selected.ebn")
```

```
EndWith
```

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

```
  .SelBackColor = 587202560
```

```
EndWith
```

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

method Appearance.Clear ()

Removes all skins in the control.

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

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

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

- borders, [MenuBarBorder](#) and [Border](#) properties
- selected item, [SelBackColor](#) property
- shadow area, [ShadowColor](#) property
- item, [BackColor](#) property
- separator item, tooltips, top level items, [Background](#) property
- and so on.

method Appearance.Remove (ID as Long)

Removes a specific skin from the control.

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

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

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

- borders, [MenuBarBorder](#) and [Border](#) properties
- selected item, [SelBackColor](#) property
- shadow area, [ShadowColor](#) property
- item, [BackColor](#) property
- separator item, tooltips, top level items, [Background](#) property
- and so on.

Control object

the Control object holds information about an inside ActiveX component. Use the [SubControl](#) property to access the Control object created using the [Add](#) method.

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.

property Control.CloseOn as CloseOnEnum

Indicates when the control is closed.

Type	Description
CloseOnEnum	A CloseOnEnum expression that indicates the way how the control is closed.

Use the CloseOn property to specify how to close an item that hosts an ActiveX control. By default, the CloseOn property is exLButtonDbIClk, and that means that the popup menu is closed if the user double clicks the ActiveX control. Use the [SelectOn](#) property to specify whether the control selects an item when user presses or releases the mouse button.

The following VB sample changes the item's caption when user double clicks a new date in a MSCal.Calendar control:

```
Private Sub ExMenu1_OleEvent(ByVal ID As Long, ByVal Ev As EXMENUlibCtl.IOleEvent)
    If Ev.Name = "DbIClick" Then
        With ExMenu1.Item(ID)
            .Caption = "Date: <b>" & .SubControl.Object.Value & "</b>"
        End With
    End If
End Sub
```

```
Private Sub Form_Load()
    With ExMenu1
        With .Items.Add("Date: <b>" & Date & "</b>", ItemTypeEnum.SubControl,
0).SubControl
            .CloseOn = exLButtonDbIClk
            .Width = 256
            .Height = 256
            .ControlID = "MSCal.Calendar"
            .Create
        End With
        .Refresh
    End With
End Sub
```

The following C++ sample changes the item's caption when user double clicks a new date in a MSCal.Calendar control:

```

#import <exmenu.dll>
#import <mscal.ocx>
void OnOleEventExmenu1(long ID, LPDISPATCH Ev)
{
    EXMENUMLib::IOleEventPtr spEvent( Ev );
    CString strName( spEvent->GetName().operator const char *() );
    if ( strName == _T("DbClick") )
    {
        CItem item = m_menu.GetItem( COleVariant( ID ) );
        MSACAL::ICalendarPtr spCalendar( item.GetSubControl().GetObject() );
        if ( spCalendar != NULL )
        {
            COleVariant vtValue;
            spCalendar->get_Value( &vtValue );
            item.SetCaption( V2S( vtValue ) );
        }
    }
}

```

The C++ sample requires `#import <exmenu.dll>` statement to include definitions for [OleEvent](#) and [OleEventParam](#) objects. The `#import <mscal.ocx>` statement imports definition for all objects in the MSCAL.Calendar library.

The V2S function converts a Variant expression to a string value:

```

static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}

```

The following VB.NET sample changes the item's caption when user double clicks a new date in a MSCal.Calendar control:

```
Private Sub AxExMenu1_OleEvent(ByVal sender As Object, ByVal e As
AxEXMENUlib._IMenuEvents_OleEventEvent) Handles AxExMenu1.OleEvent
    If e.ev.Name = "DbIClick" Then
        With AxExMenu1.item(e.iD)
            .Caption = "Date: <b>" & .SubControl.Object.Value & "</b>"
        End With
    End If
End Sub
```

The following C# sample changes the item's caption when user double clicks a new date in a MSCal.Calendar control:

```
private void axExMenu1_OleEvent(object sender,
AxEXMENUlib._IMenuEvents_OleEventEvent e)
{
    if (e.ev.Name == "DbIClick")
    {
        EXMENUlib.item item = axExMenu1[e.iD];
        MSACAL.Calendar cal = item.SubControl.Object as MSACAL.Calendar;
        if (cal != null)
            item.Caption = cal.Value.ToString();
    }
}
```

the C# sample requires adding a new reference to MSCAL.Calendar library. Select the "Project\Add Reference..." and add the Microsoft Calendar Control.

The following VFP sample changes the item's caption when user double clicks a new date in a MSCal.Calendar control:

```
*** ActiveX Control Event ***
LPARAMETERS id, ev

With Ev
    if ( .Name = "DbIClick" )
        with thisform.ExMenu1.Item(id)
```

```
.Caption = .SubControl.Object.Value  
endwith  
endif  
EndWith
```

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.

The following VB sample displays the Microsoft Chart Control to a popup menu:

```
Private Sub ExMenu1_OleEvent(ByVal ID As Long, ByVal Ev As EXMENUlibCtl.IOleEvent)
    Debug.Print Ev.Name
End Sub

Private Sub Form_Load()
    With ExMenu1
        With .Items.Add("Chart", ItemTypeEnum.SubControl, 0).SubControl
            .ControlID = "MSChart20Lib.MSChart"
            .Create
        End With
        .Refresh
    End With
End Sub
```

The following VB sample displays a HTML document inside a SubControl item:

```
With ExMenu1
    With .Items.Add(" <b>HTML</b> Document ", EXMENUlibCtl.SubControl,
1).SubControl
        .CloseOn = exClick
        .Width = 196
        .Height = 134
    End With
End With
```

```
.ControlID = "htmlfile"
```

```
.Create
```

```
With .Object()
```

```
.write "<HTML><BODY>"
```

```
.write "<p>This is a <b>HTML</b> page...</p>"
```

```
.write "<ul>"
```

```
.write "<li>1 issue</li>"
```

```
.write "<li>2 issue</li>"
```

```
.write "<li>3 issue</li>"
```

```
.write "</ul>"
```

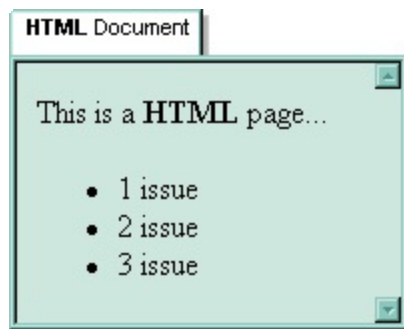
```
.write "</BODY></HTML>"
```

```
End With
```

```
End With
```

```
.Refresh
```

```
End With
```



The following VB sample adds the [Exontrol.ChartView](#) component:

```
With ExMenu1.Items
```

```
With .Add(" ActiveX ", EXMENULibCtl.ItemTypeEnum.SubControl, 1234).SubControl
```

```
.Width = 256
```

```
.Height = 256
```

```
.ControlID = "Exontrol.ChartView"
```

```
.Create
```

```
With .Object
```

```
.BeginUpdate
```

```
.BackColor = RGB(255, 255, 255)
```

```
.Appearance = 2
```

```
.HasButtons = 3
```

```
.ButtonsAlign = 0
```

```
.PenWidthLink = 3
```

```
With .Root
```

```
    .Caption = "RootSome information here.
```

```
Line 1:1
```

```
Line 2:2"
```

```
    .Image = 1
```

```
    .AddAssistant ("Assistant node")
```

```
End With
```

```
With .Nodes
```

```
    With .Add("Item 1", , "Key1")
```

```
        .HasButton = False
```

```
        .LinkTo = "Key2"
```

```
    End With
```

```
    .Add "SubItem 1", "Key1"
```

```
    With .Add("Sub Item 2", , "Key2")
```

```
        Dim s As String
```

```
        s =
```

```
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCABigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBI
```

```
        s = s +
```

```
"fFSEBhikGxSDKbgnglBgoCAAQ7F6IxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE
```

```
        .Picture = s
```

```
        .Expanded = False
```

```
        .ArrangeSiblingNodesAs = 1
```

```
    End With
```

```
    .Add "SubItem 1", "Key2"
```

```
    .Add "SubItem 2", "Key2"
```

```
End With
```

```
    .EndUpdate
```

```
End With
```

```
End With
```

```
End With
```

```
ExMenu1.Refresh
```

The following C++ sample adds the Exontrol.ChartView component:

```
#include "Item.h"
```

```

#include "Menu.h"
#include "Control.h"
#import

CItem item = m_menu.GetItems().Add( " ActiveX ", COleVariant( (long)3 /*SubControl*/ ),
COleVariant( (long)1234 ) );
CControl control = item.GetSubControl();
control.SetWidth( 256 );
control.SetHeight( 256 );
control.SetControlID( "Exontrol.ChartView" );
control.Create();
EXORGCHARTLib::IChartViewPtr spChart( control.GetObject() );
if ( spChart != NULL )
{
    spChart->BeginUpdate();
    spChart->BackColor = RGB(255, 255, 255);
    spChart->Appearance = EXORGCHARTLib::Sunken;
    spChart->HasButtons = EXORGCHARTLib::exWPlus;
    spChart->ButtonsAlign = EXORGCHARTLib::UpperLeft;
    spChart->PenWidthLink = 3;
    EXORGCHARTLib::INodePtr spRoot = spChart->Root;
    spRoot->Caption = "Root

```

Some information here.

Line 1:**1**

Line 2:2";

```

    spRoot->Image = 1;
    spRoot->AddAssistant ("Assistant node", vtMissing, vtMissing);
    EXORGCHARTLib::INodesPtr spNodes = spChart->Nodes;
    EXORGCHARTLib::INodePtr spNode1 = spNodes->Add("Item 1", vtMissing , "Key1",
vtMissing, vtMissing);
    spNode1->HasButton = false;
    spNode1->LinkTo = "Key2";
    spNodes->Add("SubItem 1", "Key1", vtMissing, vtMissing, vtMissing );
    EXORGCHARTLib::INodePtr spNode2 = spNodes->Add("Sub Item 2", vtMissing, "Key2",
vtMissing, vtMissing );
    CString s(
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCAbigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBt

```

```

);
    s = s +
"ffSEBhikGxSDKbgnglBgoCAAQ7F6lxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE

    spNode2->put_Picture( COleVariant( s ) );
    spNode2->Expanded = false;
    spNode2->ArrangeSiblingNodesAs = EXORGCHARTLib::exHorizontally;
    spNodes->Add("SubItem 1", "Key2", vtMissing, vtMissing, vtMissing );
    spNodes->Add("SubItem 2", "Key2", vtMissing, vtMissing, vtMissing );
    spChart->EndUpdate();
}
m_menu.Refresh();

```

The C++ sample requires calling the `#import <exorgchart.dll>` to import definitions for the `Exontrol.ChartView` component. It generates the `EXORGCHARTLib` namespace where you can find all objects of the `ExOrgChart` component.

The following VB.NET sample adds the `Exontrol.ChartView` component:

```

With AxExMenu1.Items
    With .Add(" ActiveX ", EXMENUMLib.ItemTypeEnum.SubControl, 1234).SubControl
        .Width = 256
        .Height = 256
        .ControlID = "Exontrol.ChartView"
        .Create()
    With .Object
        .BeginUpdate()
        .BackColor = RGB(255, 255, 255)
        .Appearance = 2
        .HasButtons = 3
        .ButtonsAlign = 0
        .PenWidthLink = 3
    With .Root
        .Caption = "Root

```

Some information here.

Line 1:1

Line 2:2"

```

        .Image = 1

```

```
.AddAssistant("Assistant node")
```

```
End With
```

```
With .Nodes
```

```
With .Add("Item 1", , "Key1")
```

```
.HasButton = False
```

```
.LinkTo = "Key2"
```

```
End With
```

```
.Add("SubItem 1", "Key1")
```

```
With .Add("Sub Item 2", , "Key2")
```

```
Dim s As String
```

```
s =
```

```
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCAbigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBl
```

```
s = s +
```

```
"fFSEBhikGxSDKbgnglBgoCAAQ7F6IxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE
```

```
.Picture = s
```

```
.Expanded = False
```

```
.ArrangeSiblingNodesAs = 1
```

```
End With
```

```
.Add("SubItem 1", "Key2")
```

```
.Add("SubItem 2", "Key2")
```

```
End With
```

```
.EndUpdate()
```

```
End With
```

```
End With
```

```
End With
```

```
AxExMenu1.CtlRefresh()
```

The following C# sample adds the Exontrol.ChartView component:

```
EXMENUMLib.Menu items = axExMenu1.Items;  
EXMENUMLib.Control control = items.Add(" ActiveX ",  
EXMENUMLib.ItemTypeEnum.SubControl, 1234).SubControl;  
control.Width = 256;  
control.Height = 256;  
control.ControlID = "Exontrol.ChartView";
```

```

control.Create();
EXORGCHARTLib.ChartView chart = control.Object as EXORGCHARTLib.ChartView;
if (chart != null)
{
    chart.BeginUpdate();
    chart.BackColor = ToUInt32(Color.White);
    chart.Appearance = EXORGCHARTLib.AppearanceEnum.Sunken;
    chart.HasButtons = EXORGCHARTLib.ExpandButtonEnum.exWPlus;
    chart.ButtonsAlign = EXORGCHARTLib.PictureDisplayEnum.UpperLeft;
    chart.PenWidthLink = 3;
    EXORGCHARTLib.Node node = chart.Root;
    node.Caption = "Root

```

Some information here.

Line 1:**1**

Line 2:2";

```

    node.Image = 1;
    node.AddAssistant ("Assistant node", null, null);
    EXORGCHARTLib.Nodes nodes = chart.Nodes;
    EXORGCHARTLib.Node node1 = nodes.Add("Item 1", null, "Key1", null, null);
    node1.HasButton = false;
    node1.LinkTo = "Key2";
    nodes.Add("SubItem 1", "Key1", "Key3", null, null);
    EXORGCHARTLib.Node node2 = nodes.Add("Sub Item 2", null, "Key2", null, null);
    String s =
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCABigbEsWGAIGA7Eo7HcbIowIpfHZQkZQKA7IspIErIBl

    s = s +
"ffSEBhikGxSDKbgnglBgoCAAQ7F6IxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE

    node2.Picture = s;
    node2.Expanded = false;
    node2.ArrangeSiblingNodesAs = EXORGCHARTLib.ArrangeSiblingEnum.exHorizontally;
    nodes.Add("SubItem 1", "Key2", "Key4", null, null);
    nodes.Add("SubItem 2", "Key2", "Key5", null, null);
    chart.EndUpdate();
}
axExMenu1.CtlRefresh();

```

The C# sample requires a new reference to the Exontrol's ExOrgChart component. Select the Project\Add Reference... and Select COM\ExOrgChart 1.0 Control Library. Once that the component is referred, the EXORGCHARTLib namespace is created, where we can find all objects and definitions for the component being inserted.

The ToUInt32 function converts a Color expression to OLE_COLOR:

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

The following VFP sample adds the Exontrol.ChartView component:

```
With thisform.ExMenu1.Items
    With .Add(" ActiveX ", 3, 1234).SubControl &&
        EXMENULibCtl.ItemTypeEnum.SubControl
            .Width = 256
            .Height = 256
            .ControlID = "Exontrol.ChartView"
            .Create
            With .Object
                .BeginUpdate
                .BackColor = RGB(255, 255, 255)
                .Appearance = 2
                .HasButtons = 3
                .ButtonsAlign = 0
                .PenWidthLink = 3
            With .Root
                .Caption = "Root"
```

Some information here.

Line 1:**1**

Line 2:2"

```
        .Image = 1
        .AddAssistant ("Assistant node")
```

EndWith

With .Nodes

With .Add("Item 1", , "Key1")

.HasButton = .f

.LinkTo = "Key2"

EndWith

.Add("SubItem 1", "Key1")

With .Add("Sub Item 2", , "Key2")

local s

s =

"gBHJJGHA5MIgAEIe4AAAFhwQiAbCAbigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBI

s = s +

"Fw2HxGJxWLxmNx2PyGRyWTymVy2XzGZzWbzmdz2f0Gh0WjzsMAAhfIEAMMf4AFmmhO

s = s +

"GlPFAkEQhEgLBYhmYgDAWBhCBsFh8HgQJCASCYafYcjQCGBAIBgKhCCwZB6kAAgFgkOBQ,

s = s +

"8TAmlYDBigMAgtAyXQyiASJzmqA4CEEf5VIAIRim8XwiiPDRzmsQUKhQLAsEqEBJhCAxSBwI

s = s +

"wAlwg8SxsAqAYHAQWggAGDgaGAKxEgETIzECOoxkqeoAgUFwiHgbQggKHhwBGAYJHIU-

s = s +

"mZh8GCBYSDoUBPjMAoansTAGBcVA4AEPINAQOAAEaDREAIYQCgWFATmgO5HI0GAvCeC

s = s +

"GKoSHAHluiGCZGASYymhgagTk8OIQjclBvCqHsNwdRAABAaOAKAwwyAtE4LAbogVnOmI

.Picture = s

.Expanded = .f

.ArrangeSiblingNodesAs = 1

EndWith

.Add("SubItem 1", "Key2")

.Add("SubItem 2", "Key2")

EndWith

.EndUpdate

EndWith

EndWith

EndWith

thisform.ExMenu1.Object.Refresh

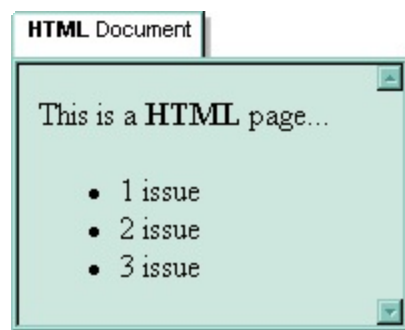
method Control.Create ()

Creates the inside 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.



The following VB sample creates a popup menu that hosts an [Exontrol.Calendar](#) component:

```
With ExMenu1
  With .Items.Add("Exontrol.<b>Calendar</b>", EXMENUlibCtl.SubControl)
    .ID = 122
    With .SubControl
      .ControlID = "Exontrol.Calendar"
      .Create
      With .Object
        .ShowWeeks = True
        .ShowTodayButton = False
      End With
    End With
  End With
```

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

The following VB sample displays a HTML document inside a SubControl item:

```
With ExMenu1
  With .Items.Add(" <b>HTML</b> Document ", EXMENUMLibCtl.SubControl,
1).SubControl
    .CloseOn = exClick
    .Width = 196
    .Height = 134
    .ControlID = "htmlfile"
    .Create
    With .Object()
      .write "<HTML> <BODY> "
      .write "<p>This is a <b>HTML</b> page...</p> "
      .write "<ul> "
        .write "<li> 1 issue</li> "
        .write "<li> 2 issue</li> "
        .write "<li> 3 issue</li> "
      .write "</ul> "
      .write "</BODY> </HTML> "
    End With
  End With
.Refresh
End With
```

The following VB sample adds the [Exontrol.ChartView](#) component:

```
With ExMenu1.Items
  With .Add(" ActiveX ", EXMENUMLibCtl.ItemTypeEnum.SubControl, 1234).SubControl
    .Width = 256
    .Height = 256
    .ControlID = "Exontrol.ChartView"
    .Create
    With .Object
      .BeginUpdate
```

.BackColor = RGB(255, 255, 255)

.Appearance = 2

.HasButtons = 3

.ButtonsAlign = 0

.PenWidthLink = 3

With .Root

.Caption = "**Root**Some information here.

Line 1:1

Line 2:2"

.Image = 1

.AddAssistant ("**Assistant** node")

End With

With .Nodes

With .Add("Item 1", , "Key1")

.HasButton = False

.LinkTo = "Key2"

End With

.Add "SubItem 1", "Key1"

With .Add("Sub Item 2", , "Key2")

Dim s As String

s =

"gBHJJGHA5MIgAEIe4AAAFhwQiAbCAbigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBl

s = s +

"fFSEBhikGxSDKbgnglBgoCAAQ7F6IxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE"

.Picture = s

.Expanded = False

.ArrangeSiblingNodesAs = 1

End With

.Add "SubItem 1", "Key2"

.Add "SubItem 2", "Key2"

End With

.EndUpdate

End With

End With

End With

The following C++ sample adds the Exontrol.ChartView component:

```
#include "Item.h"
#include "Menu.h"
#include "Control.h"
#import

CItem item = m_menu.GetItems().Add( " ActiveX ", COleVariant( (long)3 /*SubControl*/ ),
COleVariant( (long)1234 ) );
CControl control = item.GetSubControl();
control.SetWidth( 256 );
control.SetHeight( 256 );
control.SetControlID( "Exontrol.ChartView" );
control.Create();
EXORGCHARTLib::IChartViewPtr spChart( control.GetObject() );
if ( spChart != NULL )
{
    spChart->BeginUpdate();
    spChart->BackColor = RGB(255, 255, 255);
    spChart->Appearance = EXORGCHARTLib::Sunken;
    spChart->HasButtons = EXORGCHARTLib::exWPlus;
    spChart->ButtonsAlign = EXORGCHARTLib::UpperLeft;
    spChart->PenWidthLink = 3;
    EXORGCHARTLib::INodePtr spRoot = spChart->Root;
    spRoot->Caption = "Root
```

Some information here.

Line 1:**1**

Line 2:2";

```
    spRoot->Image = 1;
    spRoot->AddAssistant ( "Assistant node", vtMissing, vtMissing);
    EXORGCHARTLib::INodesPtr spNodes = spChart->Nodes;
    EXORGCHARTLib::INodePtr spNode1 = spNodes->Add("Item 1", vtMissing , "Key1",
vtMissing, vtMissing);
    spNode1->HasButton = false;
    spNode1->LinkTo = "Key2";
```

```

spNodes->Add("SubItem 1", "Key1", vtMissing, vtMissing, vtMissing );
EXORGCHARTLib::INodePtr spNode2 = spNodes->Add("Sub Item 2", vtMissing, "Key2",
vtMissing, vtMissing );
    CString s(
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCAbigbEsWGAIGA7Eo7HcbIowIpfHZQkZQKA7IspIErIBI
");
    s = s +
"ffSEBhikGxSDKbgnglBgoCAAQ7F6IxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE
";

    spNode2->put_Picture( COleVariant( s ) );
    spNode2->Expanded = false;
    spNode2->ArrangeSiblingNodesAs = EXORGCHARTLib::exHorizontally;
    spNodes->Add("SubItem 1", "Key2", vtMissing, vtMissing, vtMissing );
    spNodes->Add("SubItem 2", "Key2", vtMissing, vtMissing, vtMissing );
    spChart->EndUpdate();
}
m_menu.Refresh();

```

The C++ sample requires calling the `#import <exorgchart.dll>` to import definitions for the `Exontrol.ChartView` component. It generates the `EXORGCHARTLib` namespace where you can find all objects of the `ExOrgChart` component.

The following VB.NET sample adds the `Exontrol.ChartView` component:

```

With AxExMenu1.Items
    With .Add(" ActiveX ", EXMENUMLib.ItemTypeEnum.SubControl, 1234).SubControl
        .Width = 256
        .Height = 256
        .ControlID = "Exontrol.ChartView"
        .Create()
    With .Object
        .BeginUpdate()
        .BackColor = RGB(255, 255, 255)
        .Appearance = 2
        .HasButtons = 3
        .ButtonsAlign = 0
        .PenWidthLink = 3
    With .Root

```

```
.Caption = "Root
```

```
Some information here.
```

```
Line 1:1
```

```
Line 2:2"
```

```
.Image = 1
```

```
.AddAssistant("Assistant node")
```

```
End With
```

```
With .Nodes
```

```
With .Add("Item 1", , "Key1")
```

```
.HasButton = False
```

```
.LinkTo = "Key2"
```

```
End With
```

```
.Add("SubItem 1", "Key1")
```

```
With .Add("Sub Item 2", , "Key2")
```

```
Dim s As String
```

```
s =
```

```
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCABigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBI
```

```
s = s +
```

```
"fFSEBhikGxSDKbgnglBgoCAAQ7F6IxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE
```

```
.Picture = s
```

```
.Expanded = False
```

```
.ArrangeSiblingNodesAs = 1
```

```
End With
```

```
.Add("SubItem 1", "Key2")
```

```
.Add("SubItem 2", "Key2")
```

```
End With
```

```
.EndUpdate()
```

```
End With
```

```
End With
```

```
End With
```

```
AxExMenu1.CtlRefresh()
```

The following C# sample adds the Exontrol.ChartView component:

```
EXMENULib.Menu items = axExMenu1.Items;
```

```

EXMENUMLib.Control control = items.Add(" ActiveX ",
EXMENUMLib.ItemTypeEnum.SubControl, 1234).SubControl;
control.Width = 256;
control.Height = 256;
control.ControlID = "Exontrol.ChartView";
control.Create();
EXORGCHARTLib.ChartView chart = control.Object as EXORGCHARTLib.ChartView;
if (chart != null)
{
    chart.BeginUpdate();
    chart.BackColor = ToUInt32(Color.White);
    chart.Appearance = EXORGCHARTLib.AppearanceEnum.Sunken;
    chart.HasButtons = EXORGCHARTLib.ExpandButtonEnum.exWPlus;
    chart.ButtonsAlign = EXORGCHARTLib.PictureDisplayEnum.UpperLeft;
    chart.PenWidthLink = 3;
    EXORGCHARTLib.Node node = chart.Root;
    node.Caption = "Root

```

Some information here.

Line 1:1

Line 2:2";

```

    node.Image = 1;
    node.AddAssistant ("Assistant node", null, null);
    EXORGCHARTLib.Nodes nodes = chart.Nodes;
    EXORGCHARTLib.Node node1 = nodes.Add("Item 1", null, "Key1", null, null);
    node1.HasButton = false;
    node1.LinkTo = "Key2";
    nodes.Add("SubItem 1", "Key1", "Key3", null, null);
    EXORGCHARTLib.Node node2 = nodes.Add("Sub Item 2", null, "Key2", null, null);
    String s =
"gbHJJGHA5MIgAEIe4AAAFhwQiAbCAbigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBl

    s = s +
"fFSEBhikGxSDKbgnglBgoCAAQ7F6IxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE"

    node2.Picture = s;
    node2.Expanded = false;
    node2.ArrangeSiblingNodesAs = EXORGCHARTLib.ArrangeSiblingEnum.exHorizontally;

```

```

nodes.Add("SubItem 1", "Key2", "Key4", null, null);
nodes.Add("SubItem 2", "Key2", "Key5", null, null);
chart.EndUpdate();
}
axExMenu1.CtlRefresh();

```

The C# sample requires a new reference to the Exontrol's ExOrgChart component. Select the Project\Add Reference... and Select COM\ExOrgChart 1.0 Control Library. Once that the component is referred, the EXORGCHARTLib namespace is created, where we can find all objects and definitions for the component being inserted.

The ToUInt32 function converts a Color expression to OLE_COLOR:

```

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

```

The following VFP sample adds the Exontrol.ChartView component:

```

With thisform.ExMenu1.Items
    With .Add(" ActiveX ", 3, 1234).SubControl &&
EXMENULibCtl.ItemTypeEnum.SubControl
        .Width = 256
        .Height = 256
        .ControlID = "Exontrol.ChartView"
        .Create
    With .Object
        .BeginUpdate
        .BackColor = RGB(255, 255, 255)
        .Appearance = 2
        .HasButtons = 3
        .ButtonsAlign = 0
        .PenWidthLink = 3
    With .Root

```

.Caption = "**Root**"

Some information here.

Line 1:1

Line 2:2"

.Image = 1

.AddAssistant ("**Assistant** node")

EndWith

With .Nodes

With .Add("Item 1", , "Key1")

.HasButton = .f

.LinkTo = "Key2"

EndWith

.Add("SubItem 1", "Key1")

With .Add("Sub Item 2", , "Key2")

local s

s =

"gBHJJGHA5MIgAEIe4AAAFhwQiAbCABigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBt

s = s +

"Fw2HxGJxWLxmNx2PyGRyWTymVy2XzGZzWbzmdz2f0Gh0WjzsMAAhfIEAMMf4AFmmhO

s = s +

"GlPFAkEQhEgLBYhmYgDAWBhCBsFh8HgQJCASCYafYcjqCGBAIBgKhCCwZB6kAAgFgkOBQ,

s = s +

"8TAmlYDBigMAgtAyXQyiASJzmqA4CEEf5VIAIRim8XwiiPDRzmsQUKhQLAsEqEBJhCAxSBwl

s = s +

"wAlwg8SxsAqAYHAQWggAGDgaGAKxEgETIzECOoxkqeoAgUFwiHgbQggKHhwBGAYJHIU-

s = s +

"mZh8GCBYSDoUBPjMAoansTAGBcVA4AEPINAQOAAEaDREAIYQCgWFATmgO5HI0GAvCeC

s = s +

"GKoSHAHIuiGCZGASYymhgagTk8OIQjclBvCqHsNwdRAABAaOAKAwwyAtE4LAboqVnOml

.Picture = s

```
.Expanded = .f
```

```
.ArrangeSiblingNodesAs = 1
```

```
EndWith
```

```
.Add("SubItem 1", "Key2")
```

```
.Add("SubItem 2", "Key2")
```

```
EndWith
```

```
.EndUpdate
```

```
EndWith
```

```
EndWith
```

```
EndWith
```

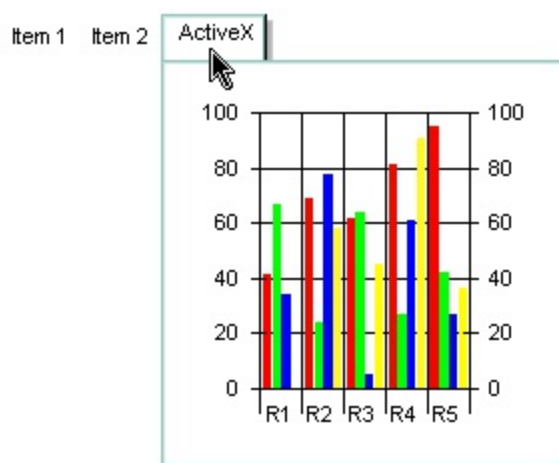
```
thisform.ExMenu1.Object.Refresh
```

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.



The following VB sample adds a Microsoft Chart Control:

```
With ExMenu1.Items
  With .Add(" ActiveX ", EXMENUlibCtl.ItemTypeEnum.SubControl, 1234).SubControl
    .Width = 200
    .Height = 200
    .ControlID = "MSChart20Lib.MSChart"
    .Create
  End With
End With
ExMenu1.Refresh
```

The following C++ sample adds a Microsoft Chart Control:

```
#include "Item.h"
#include "Menu.h"
#include "Control.h"
```

```
CItem item = m_menu.GetItems().Add( " ActiveX ", COleVariant( (long)3 /*SubControl*/ ),  
COleVariant( (long)1234 ) );  
CControl control = item.GetSubControl();  
control.SetWidth( 200 );  
control.SetHeight( 200 );  
control.SetControlID( "MSChart20Lib.MSChart" );  
control.Create();  
m_menu.Refresh();
```

The following VB.NET sample adds a Microsoft Chart Control:

```
With AxExMenu1.Items  
    With .Add(" ActiveX ", EXMENUMLib.ItemTypeEnum.SubControl, 1234).SubControl  
        .Width = 200  
        .Height = 200  
        .ControlID = "MSChart20Lib.MSChart"  
        .Create()  
    End With  
End With  
AxExMenu1.CtlRefresh()
```

The following C# sample adds a Microsoft Chart Control:

```
EXMENUMLib.Menu items = axExMenu1.Items;  
EXMENUMLib.Control control = items.Add(" ActiveX ",  
EXMENUMLib.ItemTypeEnum.SubControl, 1234).SubControl;  
control.Width = 200;  
control.Height = 200;  
control.ControlID = "MSChart20Lib.MSChart";  
control.Create();  
axExMenu1.CtlRefresh();
```

The following VFP sample adds a Microsoft Chart Control:

```
With thisform.ExMenu1.Items  
    With .Add(" ActiveX ", 3, 1234).SubControl &&  
EXMENUMLibCtl.ItemTypeEnum.SubControl  
        .Width = 200  
        .Height = 200
```

```
.ControlID = "MSChart20Lib.MSChart"
```

```
.Create
```

```
EndWith
```

```
EndWith
```

```
thisform.ExMenu1.Object.Refresh
```

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.

property Control.Object as Object

Gets the object.

Type	Description
Object	An Object created with 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 following VB sample adds the [Exontrol.ChartView](#) component:

```
With ExMenu1.Items
```

```
  With .Add(" ActiveX ", EXMENUItemLibCtl.ItemTypeEnum.SubControl, 1234).SubControl
```

```
    .Width = 256
```

```
    .Height = 256
```

```
    .ControllID = "Exontrol.ChartView"
```

```
    .Create
```

```
  With .Object
```

```
    .BeginUpdate
```

```
    .BackColor = RGB(255, 255, 255)
```

```
    .Appearance = 2
```

```
    .HasButtons = 3
```

```
    .ButtonsAlign = 0
```

```
    .PenWidthLink = 3
```

```
  With .Root
```

```
    .Caption = "RootSome information here.
```

```
Line 1:1
```

```
Line 2:2"
```

```
    .Image = 1
```

```
    .AddAssistant ("Assistant node")
```

```
  End With
```

```
With .Nodes
```

```
  With .Add("Item 1", , "Key1")
```

```
    .HasButton = False
```

```
    .LinkTo = "Key2"
```

```
  End With
```

```
  .Add "SubItem 1", "Key1"
```

```
With .Add("Sub Item 2", , "Key2")
```

```
Dim s As String
```

```
s =
```

```
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCAbigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBl
```

```
s = s +
```

```
"fFSEBhikGxSDKbgnglBgoCAAQ7F6IxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE
```

```
.Picture = s
```

```
.Expanded = False
```

```
.ArrangeSiblingNodesAs = 1
```

```
End With
```

```
.Add "SubItem 1", "Key2"
```

```
.Add "SubItem 2", "Key2"
```

```
End With
```

```
.EndUpdate
```

```
End With
```

```
End With
```

```
End With
```

```
ExMenu1.Refresh
```

The following C++ sample adds the Exontrol.ChartView component:

```
#include "Item.h"
```

```
#include "Menu.h"
```

```
#include "Control.h"
```

```
#import
```

```
CItem item = m_menu.GetItems().Add( " ActiveX ", COleVariant( (long)3 /*SubControl*/ ),  
COleVariant( (long)1234 ) );
```

```
CControl control = item.GetSubControl();
```

```
control.SetWidth( 256 );
```

```
control.SetHeight( 256 );
```

```
control.SetControlID( "Exontrol.ChartView" );
```

```
control.Create();
```

```
EXORGCHARTLib::IChartViewPtr spChart( control.GetObject() );
```

```
if ( spChart != NULL )
```

```

{
    spChart->BeginUpdate();
    spChart->BackColor = RGB(255, 255, 255);
    spChart->Appearance = EXORGCHARTLib::Sunken;
    spChart->HasButtons = EXORGCHARTLib::exWPlus;
    spChart->ButtonsAlign = EXORGCHARTLib::UpperLeft;
    spChart->PenWidthLink = 3;
    EXORGCHARTLib::INodePtr spRoot = spChart->Root;
    spRoot->Caption = "Root

```

Some information here.

Line 1:**1**

Line 2:2";

```

    spRoot->Image = 1;
    spRoot->AddAssistant ("Assistant node", vtMissing, vtMissing);
    EXORGCHARTLib::INodesPtr spNodes = spChart->Nodes;
    EXORGCHARTLib::INodePtr spNode1 = spNodes->Add("Item 1", vtMissing , "Key1",
vtMissing, vtMissing);
    spNode1->HasButton = false;
    spNode1->LinkTo = "Key2";
    spNodes->Add("SubItem 1", "Key1", vtMissing, vtMissing, vtMissing );
    EXORGCHARTLib::INodePtr spNode2 = spNodes->Add("Sub Item 2", vtMissing, "Key2",
vtMissing, vtMissing );
    CString s(
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCABigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBI
");
    s = s +
"ffSEBhikGxSDKbgnglBgoCAAQ7F6lXoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE"

    spNode2->put_Picture( COleVariant( s ) );
    spNode2->Expanded = false;
    spNode2->ArrangeSiblingNodesAs = EXORGCHARTLib::exHorizontally;
    spNodes->Add("SubItem 1", "Key2", vtMissing, vtMissing, vtMissing );
    spNodes->Add("SubItem 2", "Key2", vtMissing, vtMissing, vtMissing );
    spChart->EndUpdate();
}
m_menu.Refresh();

```

The C++ sample requires calling the `#import <exorgchart.dll>` to import definitions for the `Exontrol.ChartView` component. It generates the `EXORGCHARTLib` namespace where you can find all objects of the `ExOrgChart` component.

The following VB.NET sample adds the `Exontrol.ChartView` component:

```
With AxExMenu1.Items
```

```
    With .Add(" ActiveX ", EXMENUEnum.SubControl, 1234).SubControl
```

```
        .Width = 256
```

```
        .Height = 256
```

```
        .ControlID = "Exontrol.ChartView"
```

```
        .Create()
```

```
    With .Object
```

```
        .BeginUpdate()
```

```
        .BackColor = RGB(255, 255, 255)
```

```
        .Appearance = 2
```

```
        .HasButtons = 3
```

```
        .ButtonsAlign = 0
```

```
        .PenWidthLink = 3
```

```
    With .Root
```

```
        .Caption = "Root"
```

```
Some information here.
```

```
Line 1:1
```

```
Line 2:2"
```

```
        .Image = 1
```

```
        .AddAssistant("Assistant node")
```

```
    End With
```

```
    With .Nodes
```

```
        With .Add("Item 1", , "Key1")
```

```
            .HasButton = False
```

```
            .LinkTo = "Key2"
```

```
        End With
```

```
        .Add("SubItem 1", "Key1")
```

```
        With .Add("Sub Item 2", , "Key2")
```

```
            Dim s As String
```

```
            s =
```

```
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCAbigbEsWGAIGA7Eo7HcbIowIpfHZQkZQKA7IspIerIBI
```

```
s = s +
```

```
"fFSEBhikGxSDKbgnglBgoCAAQ7F6lxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE"
```

```
.Picture = s
```

```
.Expanded = False
```

```
.ArrangeSiblingNodesAs = 1
```

```
End With
```

```
.Add("SubItem 1", "Key2")
```

```
.Add("SubItem 2", "Key2")
```

```
End With
```

```
.EndUpdate()
```

```
End With
```

```
End With
```

```
End With
```

```
AxExMenu1.CtlRefresh()
```

The following C# sample adds the Exontrol.ChartView component:

```
EXMENULib.Menu items = axExMenu1.Items;
```

```
EXMENULib.Control control = items.Add(" ActiveX ",
```

```
EXMENULib.ItemTypeEnum.SubControl, 1234).SubControl;
```

```
control.Width = 256;
```

```
control.Height = 256;
```

```
control.ControlID = "Exontrol.ChartView";
```

```
control.Create();
```

```
EXORGCHARTLib.ChartView chart = control.Object as EXORGCHARTLib.ChartView;
```

```
if (chart != null)
```

```
{
```

```
    chart.BeginUpdate();
```

```
    chart.BackColor = ToUInt32(Color.White);
```

```
    chart.Appearance = EXORGCHARTLib.AppearanceEnum.Sunken;
```

```
    chart.HasButtons = EXORGCHARTLib.ExpandButtonEnum.exWPlus;
```

```
    chart.ButtonsAlign = EXORGCHARTLib.PictureDisplayEnum.UpperLeft;
```

```
    chart.PenWidthLink = 3;
```

```
    EXORGCHARTLib.Node node = chart.Root;
```

```
    node.Caption = "Root
```

```
Some information here.
```

Line 1:1

Line 2:2";

```
node.Image = 1;
node.AddAssistant ("Assistant node", null, null);
EXORGCHARTLib.Nodes nodes = chart.Nodes;
EXORGCHARTLib.Node node1 = nodes.Add("Item 1", null, "Key1", null, null);
node1.HasButton = false;
node1.LinkTo = "Key2";
nodes.Add("SubItem 1", "Key1", "Key3", null, null);
EXORGCHARTLib.Node node2 = nodes.Add("Sub Item 2", null, "Key2", null, null);
String s =
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCAbigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBI

s = s +
"ffSEBhikGxSDKbgnglBgoCAAQ7F6IxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE

node2.Picture = s;
node2.Expanded = false;
node2.ArrangeSiblingNodesAs = EXORGCHARTLib.ArrangeSiblingEnum.exHorizontally;
nodes.Add("SubItem 1", "Key2", "Key4", null, null);
nodes.Add("SubItem 2", "Key2", "Key5", null, null);
chart.EndUpdate();
}
axExMenu1.CtlRefresh();
```

The C# sample requires a new reference to the Exontrol's ExOrgChart component. Select the Project\Add Reference... and Select COM\ExOrgChart 1.0 Control Library. Once that the component is referred, the EXORGCHARTLib namespace is created, where we can find all objects and definitions for the component being inserted.

The ToUInt32 function converts a Color expression to OLE_COLOR:

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

```
return Convert.ToUInt32(i);
```

```
}
```

The following VFP sample adds the Exontrol.ChartView component:

```
With thisform.ExMenu1.Items
  With .Add(" ActiveX ", 3, 1234).SubControl &&
  EXMENULibCtl.ItemTypeEnum.SubControl
    .Width = 256
    .Height = 256
    .ControlID = "Exontrol.ChartView"
    .Create
  With .Object
    .BeginUpdate
    .BackColor = RGB(255, 255, 255)
    .Appearance = 2
    .HasButtons = 3
    .ButtonsAlign = 0
    .PenWidthLink = 3
  With .Root
    .Caption = "Root
```

Some information here.

Line 1:1

Line 2:2"

```
    .Image = 1
    .AddAssistant ("Assistant node")
```

EndWith

With .Nodes

```
  With .Add("Item 1", , "Key1")
```

```
    .HasButton = .f
```

```
    .LinkTo = "Key2"
```

EndWith

```
  .Add("SubItem 1", "Key1")
```

```
  With .Add("Sub Item 2", , "Key2")
```

```
    local s
```

```
    s =
```

```
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCABigbEsWGAIGA7Eo7HcbIowIpfHZQkZQKA7IspIerIBI
```

s = s +

"Fw2HxGJxWLxmNx2PyGRyWTymVy2XzGZzWbzmdz2f0Gh0WjzsMAAhfIEAMMf4AFmmhO

s = s +

"GlPFAkEQhEgLBYhmYgDAWBhCBsFh8HgQJCASCYaFYcjQGBAIBgKhCCwZB6kAAgFgkOBQ,

s = s +

"8TAmlYDBigMAgtAyXQyiASJzmqA4CEEf5VIAIRim8XwiiPDRzmsQUKhQLAsEqEBJhCAxSBwI

s = s +

"wAlwg8SxsAqAYHAQWggAGDgaGAKxEgETIzECOoxkqeoAgUFwiHgbQggKHhwBGAYJHIU4

s = s +

"mZh8GCBYSDoUBPjMAoansTAGBcVA4AEPINAQOAAEaDREAIYQCgWFATmgO5HI0GAvCeC

s = s +

"GKoSHAHluiGCZGASYymhgagTk8OIQjclBvCqHsNwdRAABAaOAKAwwyAtE4LAbogVnOmI

.Picture = s

.Expanded = .f

.ArrangeSiblingNodesAs = 1

EndWith

.Add("SubItem 1", "Key2")

.Add("SubItem 2", "Key2")

EndWith

.EndUpdate

EndWith

EndWith

EndWith

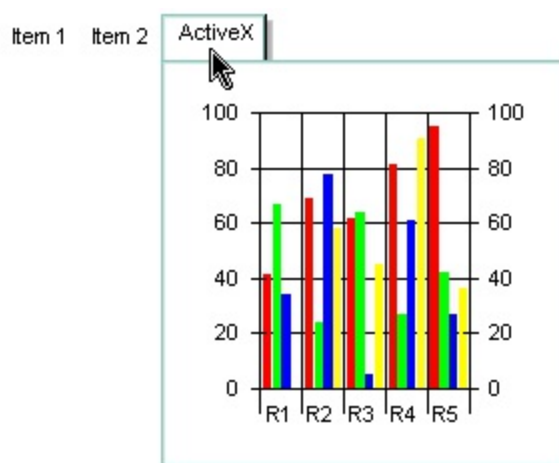
thisform.ExMenu1.Object.Refresh

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.



The following VB sample adds a Microsoft Chart Control:

```
With ExMenu1.Items
  With .Add(" ActiveX ", EXMENUlibCtl.ItemTypeEnum.SubControl, 1234).SubControl
    .Width = 200
    .Height = 200
    .ControlID = "MSChart20Lib.MSChart"
    .Create
  End With
End With
ExMenu1.Refresh
```

The following C++ sample adds a Microsoft Chart Control:

```
#include "Item.h"
#include "Menu.h"
#include "Control.h"
```

```
Cltem item = m_menu.GetItems().Add( " ActiveX ", COleVariant( (long)3 /*SubControl*/ ),  
COleVariant( (long)1234 ) );  
CControl control = item.GetSubControl();  
control.SetWidth( 200 );  
control.SetHeight( 200 );  
control.SetControlID( "MSChart20Lib.MSChart" );  
control.Create();  
m_menu.Refresh();
```

The following VB.NET sample adds a Microsoft Chart Control:

```
With AxExMenu1.Items  
    With .Add(" ActiveX ", EXMENUMLib.ItemTypeEnum.SubControl, 1234).SubControl  
        .Width = 200  
        .Height = 200  
        .ControlID = "MSChart20Lib.MSChart"  
        .Create()  
    End With  
End With  
AxExMenu1.CtlRefresh()
```

The following C# sample adds a Microsoft Chart Control:

```
EXMENUMLib.Menu items = axExMenu1.Items;  
EXMENUMLib.Control control = items.Add(" ActiveX ",  
EXMENUMLib.ItemTypeEnum.SubControl, 1234).SubControl;  
control.Width = 200;  
control.Height = 200;  
control.ControlID = "MSChart20Lib.MSChart";  
control.Create();  
axExMenu1.CtlRefresh();
```

The following VFP sample adds a Microsoft Chart Control:

```
With thisform.ExMenu1.Items  
    With .Add(" ActiveX ", 3, 1234).SubControl &&  
EXMENUMLibCtl.ItemTypeEnum.SubControl  
        .Width = 200  
        .Height = 200
```

```
.ControlID = "MSChart20Lib.MSChart"
```

```
.Create
```

```
EndWith
```

```
EndWith
```

```
thisform.ExMenu1.Object.Refresh
```

ExMenu 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: {7BE68958-94A9-4BCF-B556-8B31738F6FC2}. The object's program identifier is: "Exontrol.ExMenu". The /COM object module is: "ExMenu.dll"

The ExMenu component is a complete new type of control that displays and handles more than a menu for your application. The ExMenu simulates menu bar, popup menu, options button, toolbar, pick list, and more. Use the [VisualAppearance](#) property to add new skins to the control. The ExMenu supports the following properties and methods:

Name	Description
AddAccelerator	Associates an accelerator key to the menu item.
AllowChevron	Retrieves or sets a value that indicates whether the control programmatically displays the chevron or the scroll.
Appearance	Retrieves or sets a value that indicates the menu's appearance.
AttachTemplate	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
BackColor	Retrieves or sets the menu's background color.
Background	Returns or sets a value that indicates the background color for parts in the control.
Border	Retrieves or sets a value that indicates the menu's border.
ChevronImage	Retrieves or sets a value that indicates the index of icon being displayed as a chevron.
ClearAccelerators	Clears the accelerator keys collection.
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	Retrieves or sets a value that indicates whether the control is enabled or disabled.
EventParam	Retrieves or sets a value that indicates the current's event parameter.
ExecuteTemplate	Executes a template and returns the result.
Focusable	Gets or sets a value that indicates whether the control can receive focus.

Font	Retrieves or sets the menu's font.
ForeColor	Retrieves or sets the menu's foreground color.
HighLightBorderColor	Retrieves or sets a color that indicates the highlight border's color.
HTMLPicture	Adds or replaces a picture in HTML captions.
hWnd	Retrieves the menu's window handle.
Images	Sets the control's image list at runtime. The Handle should be a handle to an Image List control.
ImageSize	Retrieves or sets the size of icons the control displays..
item	Finds the item given its identifier or its name.
ItemHeight	Retrieves or sets the item's height.
Items	Retrieves a Menu object that handles adding, removing or changing items at runtime.
Load	Loads the menu saved using Save method.
MenuBarBorder	Specifies the menu bar's border.
OpenMode	Specifies the way how the menu is opened.
OpenOnClick	Sets or gets a value that indicates the way how the user opens the menus, using the mouse.
Picture	Retrieves or sets the background's picture.
PictureDisplay	Retrieves or sets a value that indicates the way how the picture is displayed.
PopupBackColor	Retrieves or sets a color that indicates the popup menu's background color.
PopupForeColor	Retrieves or sets a color that indicates the popup menu's text color.
Refresh	Refreshes the control.
RemoveAccelerator	Removes the menu item's accelerator key.
Replacelcon	Adds a new icon, replaces an icon or clears the control's image list.
Save	Saves the menu to the destination.
ScrollImage	Retrieves or sets a value that indicates the index of icon being displayed when the left or right scrolling item is visible on the menu bar.
ScrollOnDrop	Specifies the effect to show the popup menu when clicking

an item, such as scrolling, lighting up, and so on.

[ScrollOnWheel](#)

Indicates whether the menu gets scrolled once the user rotates the mouse wheel.

[SelBackColor](#)

Retrieves or sets a color that indicates the selection's background color.

[SelectOn](#)

Specifies whether the control selects an item when user presses or releases the mouse button.

[SelForeColor](#)

Retrieves or sets a color that indicates the selection's text color.

[SepAcc](#)

Specifies a string expression that splits the caption and accelerator key in the item.

[ShadowColor](#)

Specifies the shadow color.

[ShowPopup](#)

Shows a popup menu.

[Template](#)

Specifies the control's template.

[TemplateDef](#)

Defines inside variables for the next Template/ExecuteTemplate call.

[TemplateResult](#)

Gets the result of the last Template call.

[TemplateResultN](#)

Gets the result of the last Template call, as double.

[TemplateResultS](#)

Gets the result of the last Template call, as string.

[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.

[Version](#)

Retrieves the control's version.

[Visibility](#)

Specify the popup's visibility in percents: 90% is barely visible, and 10% is nearly opaque.

[VisualAppearance](#)

Retrieves the control's appearance.

method ExMenu.AddAcelerator (ID as Long, KeyCode as Integer, [CtrlKey as Variant], [ShiftKey as Variant], [AltKey as Variant])

Associates an accelerator key to the menu item.

Type	Description
ID as Long	A long expression that specifies the identifier of the item being activated when accelerator key is pressed.
KeyCode as Integer	A long expression that defines the code of the key being the item's accelerator
CtrlKey as Variant	A boolean expression that indicates whether the KeyCode key and CTRL key defines the accelerator
ShiftKey as Variant	A boolean expression that indicates whether the KeyCode key and SHIFT key defines the accelerator
AltKey as Variant	A boolean expression that indicates whether the KeyCode key and ALT key defines the accelerator

Use the AddAcelerator method to add accelerators to the menu. The AddAccelerator associates a key to an item. If the user presses one of the control's accelerator keys it fires the [Select](#) event. The Select event is not fired if the user presses an accelerator key that's associated to a disabled or a hidden item. Use the [Visible](#) property to hide an item. Use the [Enabled](#) property to disable an item. Use the form's KeyDown event to get the code of the key you press. Use the [Add](#) method to add new items.

The following VB sample assigns the CTRL+F accelerator to an item:

```
With ExMenu1
  With .Items
    With .Add("Popup", EXMENUMLibCtl.SubMenu, 1221).SubMenu
      .Add "File  CTRL+F", EXMENUMLibCtl.Default, 1222
    End With
  End With
  .AddAcelerator 1222, KeyCodeConstants.vbKeyF, True, False, False
  .Refresh
End With
```

If you run the sample, and press CTRL+F combination, the [Select](#) event is fired.

The following C++ sample assigns the CTRL+F accelerator to an item:

```

COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
CItem item = m_menu.GetItems().Add( "Popup", COleVariant( long(2) /*SubMenu*/),
COleVariant(long(1221)));
CItem subItem = item.GetSubMenu().Add( "File  CTRL+F", COleVariant( long(0)
/*Default*/), COleVariant(long(1222)) );
m_menu.AddAccelerator( 1222, 70, COleVariant( VARIANT_TRUE ), COleVariant(
VARIANT_FALSE ), COleVariant( VARIANT_FALSE ) );
m_menu.Refresh();

```

The following VB.NET sample assigns the CTRL+F accelerator to an item:

```

With AxExMenu1
  With .Items
    With .Add("Popup", EXMENUMLib.ItemTypeEnum.SubMenu, 1221).SubMenu
      .Add("File  CTRL+F", EXMENUMLib.ItemTypeEnum.Default, 1222)
    End With
  End With
  .AddAccelerator(1222, Convert.ToInt32(Keys.F), True, False, False)
  .CtlRefresh()
End With

```

The following C# sample assigns the CTRL+F accelerator to an item:

```

EXMENUMLib.Menu menu = axExMenu1.Items.Add("Popup",
EXMENUMLib.ItemTypeEnum.SubMenu, 1221).SubMenu;
menu.Add("File  CTRL+F", EXMENUMLib.ItemTypeEnum.Default, 1222);
axExMenu1.AddAccelerator(1222, Convert.ToInt16(Keys.F), true, false, false);
axExMenu1.CtlRefresh();

```

The following VFP sample assigns the CTRL+F accelerator to an item:

```

With thisform.ExMenu1
  With .Items
    With .Add("Popup", 2, 1221).SubMenu
      .Add("File  CTRL+F", 0, 1222)
    EndWith
  EndWith
  .AddAccelerator(1222, 70, .t., .f., .f)

```

.Object.Refresh
EndWith

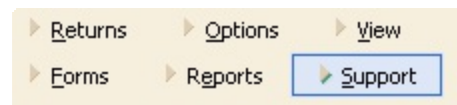
property ExMenu.AllowChevron as ChevronEnum

Retrieves or sets a value that indicates whether the control programmatically displays the chevron or scroll items on the menu bar.

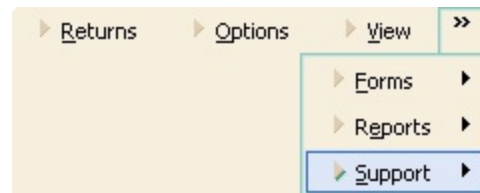
Type	Description
ChevronEnum	A ChevronEnum expression that indicates whether the control displays chevron or scroll items to the menu bar.

By default, the AllowChevron property is exChevron (True). The chevron/scroll gets displayed when the items on the menu bar do not fit the menu's client area. Use the [ChevronImage](#) property to replace the image for a chevron, when AllowChevron property is exChevron. Use the [ScrollImage](#) property to assign custom icons to scrolling items, when the AllowChevron property is exScroll. Use the [Images](#) method to assign icons to the control. Use the [Image](#) property to assign an icon to an item.

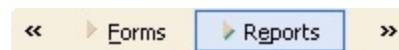
The following screen shot shows the control when AllowChevron property is exDefault.



The following screen shot shows the control when AllowChevron property is exChevron.



The following screen shot shows the control when AllowChevron property is exScroll.



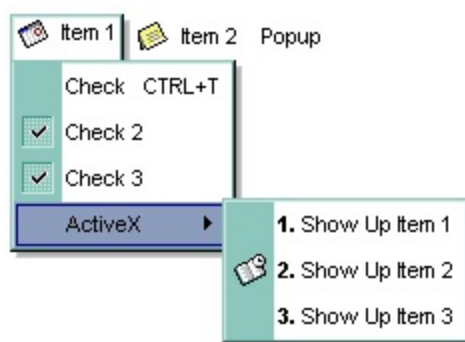
property ExMenu.Appearance as AppearanceEnum

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

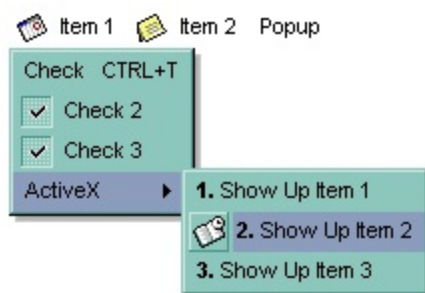
Type	Description
AppearanceEnum	An AppearanceEnum expression that indicates the control's appearance.

Use the Appearance property to change the control's appearance. Use the [Border](#) property to define the control's borders. Use the [Font](#) property to specify the control's font. Use the [BackColor](#) property to specify the control's background color. Use the [ForeColor](#) property to specify the control's foreground color. Use the [MenuBarBorder](#) property to specify the border of the control. Use the [PopupBackColor](#) property to specify the background color for the drop down menus. Use the [PopupForeColor](#) property to specify the foreground color for the drop down menus. Use the [ShadowColor](#) property to specify the color used to paint the shadow on the left side of the drop down menu, when Appearance property is Flat. Use the [Visibility](#) property to specify the drop down visibility. Use the [ItemHeight](#) property to specify the height for all items.

The following screen shot shows the control when Appearance property is Flat:



The following screen shot shows the control when Appearance property is Normal:



The following screen shot shows the control when Appearance property is Button:

Item 1 Item 2 Popup

- Check CTRL+T
- Check 2
- Check 3
- ActiveX ▶
 - 1. Show Up Item 1
 - 2. Show Up Item 2
 - 3. Show Up Item 3

method ExMenu.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 ExMenu1_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 ExMenu.BackColor as Color

Retrieves or sets the menu's background color.

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

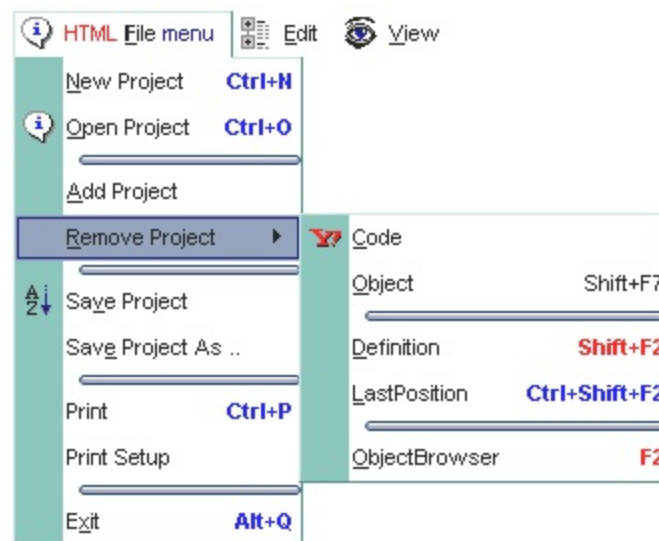
Use the [ForeColor](#) and [BackColor](#) properties to customize the menu's colors. Use the [PopupForeColor](#) property to change the foreground color for the popup menu. Use the [PopupBackColor](#) property to change the background color for the popup menu. Use the [UseBackColor](#), [BackColor](#), [UseForeColor](#), [ForeColor](#) properties to define colors for any menu item. Use the [SelBackColor](#) and [SelForeColor](#) properties to define colors for selected item. Use the [ShadowColor](#) property to specify the color used to paint the shadow border on the left side of the drop down menu, when the [Appearance](#) property is Flat. Use the [Picture](#) property to load a picture on the control's background.


property ExMenu.Background(Part as BackgroundPartEnum) as Color

Returns or sets a value that indicates the background color for parts in the control.

Type	Description
Part as BackgroundPartEnum	A BackgroundPartEnum expression that indicates a part in the control.
Color	A Color expression that indicates the background color for a specified part. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

The Background property specifies a background color or a visual appearance for specific parts in the control. If the Background property is 0, the control draws the part as default. Use the [Add](#) method to add new skins to the control. Use the [Remove](#) method to remove a specific skin from the control. Use the [Clear](#) method to remove all skins in the control.



The following VB sample changes the appearance for the separator item. The sample uses the "  " skin.

With ExMenu1

```
.VisualAppearance.Add 4, "D:\Temp\ExMenu.Help\separator.ebn"
```

```
.Background(exSeparatorItem) = &H4000000
```

End With

The following C++ sample changes the appearance for the separator item:

```
#include "Appearance.h"  
m_menu.GetVisualAppearance().Add( 4, COleVariant(  
"D:\\Temp\\ExMenu.Help\\separator.ebn" ) );  
m_menu.SetBackground( 0, 0x4000000 );
```

The following VB.NET sample changes the appearance for the separator item:

```
With AxExMenu1  
    .VisualAppearance.Add(4, "D:\\Temp\\ExMenu.Help\\separator.ebn")  
    .set_Background(EXMENU.Lib.BackgroundPartEnum.exSeparatorItem, &H4000000)  
End With
```

The following C# sample changes the appearance for the separator item:

```
axExMenu1.VisualAppearance.Add(4, "D:\\Temp\\ExMenu.Help\\separator.ebn");  
axExMenu1.set_Background(EXMENU.Lib.BackgroundPartEnum.exSeparatorItem,  
0x4000000);
```

The following VFP sample changes the appearance for the separator item:

```
with thisform.ExMenu1  
    .VisualAppearance.Add(4, "D:\\Temp\\ExMenu.Help\\separator.ebn")  
    .Background(0) = 67108864  
endwith
```

where the 67108864 in hexa is 0x4000000

property ExMenu.Border as BorderEnum

Retrieves or sets a value that indicates the menu's border.

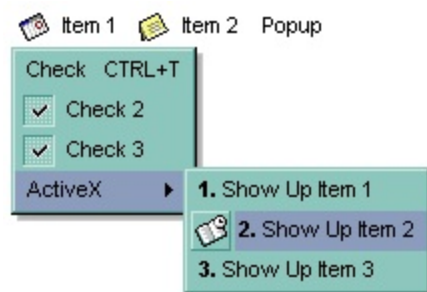
Type

Description

[BorderEnum](#)

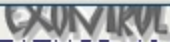
A BorderEnum expression that indicates the menu's border (it specifies the appearance of the borders for the drop down menus). Or a color expression whose last 7 bits in the high significant byte of the value indicates the index of the skin in the [Appearance](#) collection, being displayed as control's borders. For instance, if the Appearance = 0x1000000, indicates that the first skin object in the Appearance collection defines the control's border. ***The Client object in the skin, defines the client area of the control. The items are always shown in the control's client area. The skin may contain transparent objects, and so you can define round corners. The [frame.ebn](#) file contains such of objects. Use the [eXButton's Skin builder](#) to view or change this file***

By default, the Border property is ShadowBorder. Use the Border property to define the menu's border. Use the [MenuBarBorder](#) property to define the border of the menu bar. The menu bar is displayed into the form. Use the [Appearance](#) property to define the menu's appearance. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips.



Item 1

This is a bit of text that's shown when the cursor hovers the **item**.

logo **inside ...**  ▶

Subitem 2

April 2007

Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

property ExMenu.ChevronImage as Long

Retrieves or sets a value that indicates the index of icon being displayed as a chevron.

Type	Description
Long	A long expression that indicates the index of icon being displayed as a chevron.

By default, the ChevronImage property is -1. In this case, the control uses the default picture for a chevron. Use the ChevronImage property to assign a custom picture to the chevron. The index of the icon being displayed as a chevron should be zero based. Use the [AllowChevron](#) property to specify whether the control automatically displays a chevron/scroll when items on the menu bar do not fit the menu's client area. Use the [Images](#) method to assign icons to the control at runtime.

method `ExMenu.ClearAccelerators ()`

Clears the accelerator keys collection.

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

Use the [RemoveAccelerator](#) method to remove a particular accelerator. Use the [AddAccelerator](#) method to assign an accelerator key to an item. Use the [Remove](#) method to remove an item. Use the [Visible](#) property to hide an item. Use the [Enabled](#) property to disable an item. The item's accelerator is disabled if the item is disabled or it is hidden.

property ExMenu.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 indicates a predefined value listed below, a string expression that indicates the path to a cursor file, a long expression that indicates the handle of the cursor.

Use the Cursor property to specify the cursor that control displays when the mouse pointer hovers the item. Use the [Cursor](#) property to specify the shape of the cursor when the mouse pointer hovers an item. Use the [Cursor](#) property to specify the shape of the cursor when the mouse pointer hovers a sub menu.

Here's the list of predefined values (string expressions):

- **"exDefault"** - (Default) Shape determined by the object.
- **"exArrow"** - Arrow.
- **"exCross"** - Cross (cross-hair pointer).
- **"exIBeam"** - I Beam.
- **"exIcon"** - Icon (small square within a square).
- **"exSize"** - Size (four-pointed arrow pointing north, south, east, and west).
- **"exSizeNESW"** - Size NE SW (double arrow pointing northeast and southwest).
- **"exSizeNS"** - Size N S (double arrow pointing north and south).
- **"exSizeNWSE"** - Size NW, SE.
- **"exSizeWE"** - Size W E (double arrow pointing west and east).
- **"exUpArrow"** - Up Arrow.
- **"exHourglass"** - Hourglass (wait).
- **"exNoDrop"** - No Drop.
- **"exArrowHourglass"** - Arrow and hourglass.
- **"exHelp"** - Arrow and question mark.
- **"exSizeAll"** - Size all.

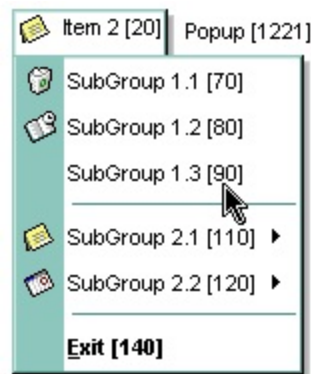
If the cursor value is a string expression, the control looks first if it is not a predefined value like listed above, and if not, it tries to load the cursor from a file. If the Cursor property is a long expression it always indicates a handle to a cursor. The API functions like: LoadCursor or LoadCursorFromFile retrieves a handle to a cursor. In .NET framework, the Handle parameter of the Cursor object specifies the handle to the cursor. Use the Cursors object to access to the list of predefined cursors in the .NET framework.

property ExMenu.Debug as Boolean

Retrieves or sets a value that indicating whether the item's identifier is visible.

Type	Description
Boolean	A boolean expression that indicates whether the control displays the item's identifier.

Use the Debug property to display item identifiers. The property is hidden, and that's possible that your browser will not show it. Use the [ID](#) property to identify an item. The ID of the item property is displayed between [] brackets. Use the [Caption](#) property to specify the caption of the item.



The following VB sample displays the identifiers for all items:

```
ExMenu1.Debug = True
```

The following C++ sample displays the identifiers for all items:

```
static BYTE parms[] = VTS_BOOL;  
m_menu.InvokeHelper(0xE, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms, TRUE );
```

The following VB.NET sample displays the identifiers for all items:

```
With AxExMenu1  
    .Debug = True  
End With
```

The following C# sample displays the identifiers for all items:

```
axExMenu1.Debug = true;
```

The following VFP sample displays the identifiers for all items:

With thisform.ExMenu1

.Debug = .t.

EndWith

property ExMenu.Enabled as Boolean

Retrieves or sets a value that indicates whether the control is enabled or disabled.

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

Use the Enabled property to disable the control. The Enabled property does not change the control's appearance. Use the [ForeColor](#) property to specify the control's foreground color. Use the [BackColor](#) property to specify the control's background color. Use the [Font](#) property to specify the control's font. Use the [Enabled](#) property to disable an item. Use the [Visible](#) property to hide or show an item.

property ExMenu.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 ExMenu.ExecuteTemplate (Template as String)

Executes a template and returns the result.

Type	Description
Template as String	A Template string being executed
Return	Description
Variant	A Variant expression that indicates the result after executing the Template.

Use the ExecuteTemplate property to returns the result of executing a template file. Use the [Template](#) property to execute a template without returning any result. Use the ExecuteTemplate property to execute code by passing instructions as a string (template string).

For instance, the following sample displays the control's background color:

```
Debug.Print ExMenu1.ExecuteTemplate("BackColor")
```

Most of our UI components provide a Template page that's accessible in design mode. No matter what programming language you are using, you can have a quick view of the component's features using the WYSWYG Template editor.

- Place the control to your form or dialog.
- Locate the Properties item, in the control's context menu, in design mode. If your environment doesn't provide a Properties item in the control's context menu, please try to locate in the Properties browser.
- Click it, and locate the Template page.
- Click the Help button. In the left side, you will see the component, in the right side, you will see a x-script code that calls methods and properties of the control.

The control's Template page helps user to initialize the control's look and feel in design mode, using the x-script language that's easy and powerful. The Template page displays the control on the left side of the page. On the right side of the Template page, a simple editor is displayed where user writes the initialization code. The control's look and feel is automatically updated as soon as the user types new instructions. The Template script is saved to the container persistence (when Apply button is pressed), and it is executed when the control is initialized at runtime. Any component that provides a WYSWYG Template page, provides a Template property. The Template property executes code from a string (template string).

The Template script is composed by lines of instructions. Instructions are separated by "\n\r" (newline) characters.

An instruction can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- **variable = property(list of arguments)** *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: h = InsertItem(0,"New Child"))*
- **property(list of arguments) = value** *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- **method(list of arguments)** *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- **{** *Beginning the object's context. The properties or methods called between { and } are related to the last object returned by the property prior to { declaration.*
- **}** *Ending the object's context*
- **object. property(list of arguments).property(list of arguments)....** *The .(dot) character splits the object from its property. For instance, the Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.*

The Template supports the following general functions:

- **RGB(R,G,B)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **CreateObject(progID)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

property `ExMenu.Focusable` as Boolean

Gets or sets a value that indicates whether the control can receive focus.

Type	Description
Boolean	A Boolean expression that specifies whether the menu receives the focus when the user clicks it.

By default, the `Focusable` property is `True`, which means that the control gains the focus once the user clicks the menu. Use the `Focusable` property on `False`, to prevent receiving the focus, when the user clicks the menu. Use the `Focusable` property on `False`, when you need to assign the menu to a MDI frame, or whether the menu requires no focusing or activation. The menu may be activated if it contains edit controls or ActiveX controls.

property ExMenu.Font as IFontDisp

Retrieves or sets the menu's font.

Type	Description
IFontDisp	A Font object that defines the menu's font.

Use the Font property to specify the control's font. Use the properties like [Bold](#), [Italic](#), [Underline](#), [Strikeout](#) to apply different font attribute for any item in the menu. Use the [Caption](#) property to specify an HTML caption for an item. Use the [ItemHeight](#) property to specify the height for all items. Use the [Refresh](#) method to refresh the control.

The following VB sample assigns by code a new font to the control:

```
With ExMenu1
  With .Font
    .Name = "Tahoma"
  End With
  .Refresh
End With
```

The following C++ sample assigns by code a new font to the control:

```
COleFont font = m_menu.GetFont();
font.SetName( "Tahoma" );
m_menu.Refresh();
```

the C++ sample requires definition of COleFont class (`#include "Font.h"`)

The following VB.NET sample assigns by code a new font to the control:

```
With AxExMenu1
  Dim font As System.Drawing.Font = New System.Drawing.Font("Tahoma", 10,
  FontStyle.Regular, GraphicsUnit.Point)
  .Font = font
  .CtlRefresh()
End With
```

The following C# sample assigns by code a new font to the control:

```
System.Drawing.Font font = new System.Drawing.Font("Tahoma", 10, FontStyle.Regular);
```

```
axExMenu1.Font = font;  
axExMenu1.CtlRefresh();
```

The following VFP sample assigns by code a new font to the control:

```
with thisform.ExMenu1.Object  
  .Font.Name = "Tahoma"  
  .Refresh()  
endwith
```

property ExMenu.ForeColor as Color

Retrieves or sets the menu's foreground color.

Type	Description
Color	A color expression that indicates the the menu's foreground color.

Use the ForeColor and [BackColor](#) properties to customize the menu's colors. Use the [PopupForeColor](#) property to change the foreground color for the popup menu. Use the [PopupBackColor](#) property to change the background color for the popup menu. Use the [UseBackColor](#), [BackColor](#), [UseForeColor](#), [ForeColor](#) properties to define colors for any menu item. Use the [SelBackColor](#) and [SelForeColor](#) properties to define colors for selected item.

property ExMenu.HighlightBorderColor as Color

Retrieves or sets a color that indicates the highlight border 's color.

Type	Description
Color	A color expression that indicates the highlight border 's color.

The control displays a highlight border around the selected item, when the menu's [Appearance](#) property is Flat. Use the [SelBackColor](#) and [SelForeColor](#) properties to define colors for selected item. The control fires the [Select](#) event when the user selects an item. Use the [ShadowColor](#) property to specify the color used to paint the shadow on the left side of the drop down menu, when Appearance property is Flat. Use the [ItemHeight](#) property to specify the height for all items. Use the [Border](#) property to define the control's borders. Use the [Font](#) property to specify the control's font. Use the [BackColor](#) property to specify the control's background color. Use the [ForeColor](#) property to specify the control's foreground color.



property ExMenu.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),

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). Use the [Caption](#) property to specify the item's caption.

property ExMenu.hWnd as Long

Retrieves the menu'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 ExMenu.Images (Handle as Variant)

Sets the control's image list at runtime. The Handle should be a handle to an Image List control.

Type

Description

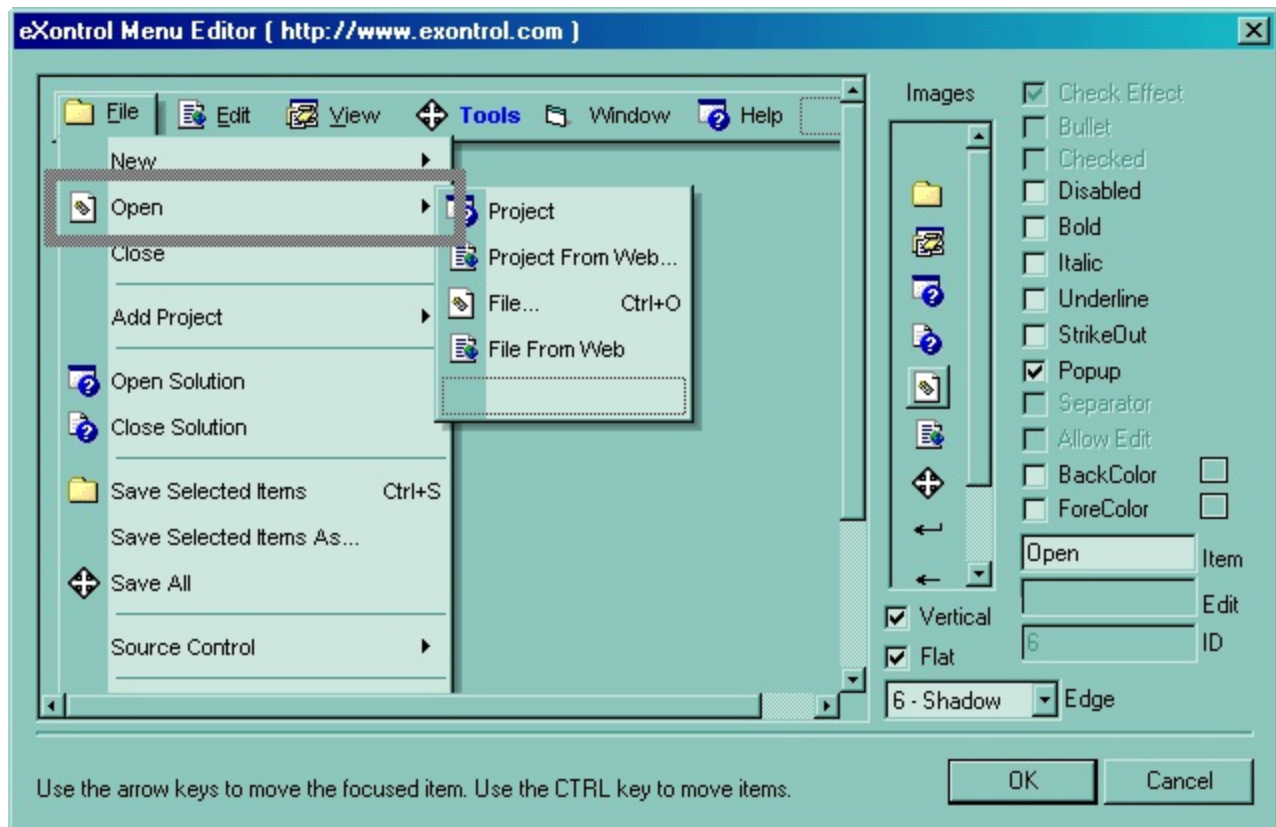
The Handle parameter can be:

- A string expression that specifies the ICO file to add. The ICO file format is an image file format for computer icons in Microsoft Windows. ICO files contain one or more small images at multiple sizes and color depths, such that they may be scaled appropriately. For instance, `Images("c:\temp\copy.ico")` method adds the `sync.ico` file to the control's Images collection (*string, loads the icon using its path*)
- A string expression that indicates the BASE64 encoded string that holds the icons list. Use the Exontrol's [ExImages](#) tool to save/load your icons as BASE64 encoded format. In this case the string may begin with "gBJJ..." (*string, loads icons using base64 encoded string*)
- A reference to a Microsoft ImageList control (`mscomctl.ocx`, `MSComctlLib.ImageList` type) that holds the icons to add (*object, loads icons from a Microsoft ImageList control*)
- A reference to a Picture (IPictureDisp implementation) that holds the icon to add. For instance, the VB's `LoadPicture (Function LoadPicture([FileName], [Size], [ColorDepth], [X], [Y]) As IPictureDisp)` or `LoadResPicture (Function LoadResPicture(id, restype As Integer) As IPictureDisp)` returns a picture object (*object, loads icon from a Picture object*)
- A long expression that identifies a handle to an Image List Control (the Handle should be of HIMAGELIST type). On 64-bit platforms, the Handle parameter must be a Variant of LongLong / LONG_PTR data type (signed 64-bit (8-byte) integers), saved under lVal field, as VT_I8 type. The LONGLONG / LONG_PTR is `__int64`, a 64-bit integer. For instance, in C++ you can use as `Images(COleVariant((LONG_PTR)hImageList))` or `Images(COleVariant(`

Handle as Variant

(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 insert icons at design time, by drag and drop files to images panel. Select the Properties context menu, when the control is in design mode, to display the control's editor like in the following screen shot. Use the [Replacelcon](#) property to add, remove, replace, or clear the images collection. The [ImageSize](#) property defines the size (width/height) of the icons within the control's Images collection.



The following VB sample loads icons from a Microsoft Image List control:

```
ExMenu1.Images ImageList1.hImageList
```

property ExMenu.ImageSize as Long

Retrieves or sets the size of icons the control displays..

Type	Description
Long	A long expression that defines the size of icons the control displays.

By default, the ImageSize property is 16 (pixels). The ImageSize property specifies the size of icons being loaded using the [Images](#) method. The control's Images collection is cleared if the ImageSize property is changed, so it is recommended to set the ImageSize property before calling the Images method. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. For instance, if the ICO file to load includes different types the one closest with the size specified by ImageSize property is loaded by Images method. The ImageSize property does NOT change the height for the control's font.

property ExMenu.item (ID as Variant) as Item

Finds the item given its identifier or its name.

Type	Description
ID as Variant	A long expression that indicates the item's identifier, or a string expression that indicates the item's caption searched for.
Item	An Item object that defines the menu item.

The Item property is the default property of the ExMenu ActiveX Control, so the following statements `ExMenu1.Item(1234)` and `ExMenu1(1234)` are equivalents. The Item property looks recursively for the Item that has the specified identifier or caption. Also, the `ExMenu1.Item(1234)` and `ExMenu1.Items.Find(1234)` are equivalents. The Item property looks for first item that has the given identifier or given caption. It is recommended to associate unique identifiers for the items. Use the [Debug](#) property to display items identifiers.

The following VB sample changes the foreground color for the item with the identifier 50:

```
With ExMenu1.Item(50)
    .ForeColor = vbRed
End With
```

The following C++ sample changes the foreground color for the item with the identifier 50:

```
CItem item = m_menu.GetItem( COleVariant( long( 50 ) ) );
item.SetForeColor( RGB(255,0,0) );
```

The following VB.NET sample changes the foreground color for the item with the identifier 50:

```
With AxExMenu1.item(50)
    .ForeColor = ToUInt32(Color.Red)
End With
```

where the `ToUInt32` function converts a `Color` expression to `OLE_COLOR`:

```
Shared Function ToUInt32(ByVal c As Color) As UInt32
    Dim i As Long
    i = c.R
```

```
i = i + 256 * c.G  
i = i + 256 * 256 * c.B  
ToUInt32 = Convert.ToUInt32(i)  
End Function
```

The following C# sample changes the foreground color for the item with the identifier 50:

```
EXMENUMLib.item item = axExMenu1[50];  
item.ForeColor = ToUInt32(Color.Red);
```

where the ToUInt32 function converts a Color expression to OLE_COLOR:

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

The following VFP sample changes the foreground color for the item with the identifier 50:

```
With thisform.ExMenu1.Item(50)  
    .ForeColor = RGB(255,0,0)  
EndWith
```

property ExMenu.ItemHeight(Appearance as AppearanceEnum) as Long

Retrieves or sets the item's height.

Type	Description
Appearance as AppearanceEnum	An Appearance expression that indicates the menu's appearance. The Appearance could be Normal, or Flat/Button. So, the valid values are 0 to hold the height for the item while the menu's appearance is Normal, and the 1 to hold the height of the item while the menu's appearance is Flat or Button.
Long	A long expression that indicates the height of the item, in pixels.

The ItemHeight property specifies the height for the items in the menu. Use the [Appearance](#) property to specify the control's appearance. Use the [Font](#) property to specify the control's font. Use the [Refresh](#) method to refresh the control's content.

The following VB sample changes the item's height for current appearance:

```
With ExMenu1
    .ItemHeight(.Appearance) = 18
    .Refresh
End With
```

The following C++ sample changes the item's height for current appearance:

```
m_menu.SetItemHeight( m_menu.GetAppearance(), 18 );
m_menu.Refresh();
```

The following VB.NET sample changes the item's height for current appearance:

```
With AxExMenu1
    .set_ItemHeight(.Appearance, 18)
    .CtlRefresh()
End With
```

The following C# sample changes the item's height for current appearance:

```
axExMenu1.set_ItemHeight(axExMenu1.Appearance, 18);
axExMenu1.CtlRefresh();
```

The following VFP sample changes the item's height for current appearance:

```
With thisform.ExMenu1
```

```
  .ItemHeight(.Appearance) = 18
```

```
  .Object.Refresh()
```

```
EndWith
```

property ExMenu.Items as Menu

Retrieves a Menu object that handles adding, removing or changing items at runtime.

Type	Description
Menu	A Menu object that helps to add, remove, change menu items at runtime.

Use the Items property to access the menu items collection. Use the [SubMenu](#) property to access the sub items of a popup menu. Use the [Add](#) method to insert new items to the menu. Use the [AddAccelerator](#) method to associate accelerator keys to the items.

The following VB sample adds some items that are aligned to the right:

```
With ExMenu1
    .Border = BumpBorder
    With .Items
        With .Add("Menu 1", EXMENU.LibCtl.SubMenu).SubMenu
            .Add("File").Alignment = exRight
            .Add("Open").Alignment = exRight
            .Add ("Print Preview")
        End With
    End With
    .Refresh
End With
```

The following C++ sample adds some items that are aligned to the right:

```
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
m_menu.SetBorder( 5/*BumpBorder*/ );
CItem item = m_menu.GetItems().Add( "Menu 1", COleVariant( (long)2 /*SubMenu*/ ),
vtMissing );
CItem itemF = item.GetSubMenu().Add( "File", vtMissing, vtMissing );
itemF.SetAlignment( 2 /*RightAlignment*/ );
CItem itemO = item.GetSubMenu().Add( "Open", vtMissing, vtMissing );
itemO.SetAlignment( 2 /*RightAlignment*/ );
item.GetSubMenu().Add("Print Preview", vtMissing, vtMissing);
m_menu.Refresh();
```

The following VB.NET sample adds some items that are aligned to the right:

```
With AxExMenu1
```

```
.Border = EXMENUMLib.BorderEnum.BumpBorder
```

```
With .Items
```

```
With .Add("Menu 1", EXMENUMLib.ItemTypeEnum.SubMenu).SubMenu
```

```
.Add("File").Alignment = EXMENUMLib.AlignmentEnum.exRight
```

```
.Add("Open").Alignment = EXMENUMLib.AlignmentEnum.exRight
```

```
.Add("Print Preview")
```

```
End With
```

```
End With
```

```
.CtlRefresh()
```

```
End With
```

The following C# sample adds some items that are aligned to the right:

```
axExMenu1.Border = EXMENUMLib.BorderEnum.BumpBorder;
```

```
EXMENUMLib.Menu items = axExMenu1.Items;
```

```
EXMENUMLib.Menu subMenu = items.Add("Menu 1", EXMENUMLib.ItemTypeEnum.SubMenu,  
null).SubMenu;
```

```
subMenu.Add("File", null, null).Alignment = EXMENUMLib.AlignmentEnum.exRight;
```

```
subMenu.Add("Open", null, null).Alignment = EXMENUMLib.AlignmentEnum.exRight;
```

```
subMenu.Add("Print Preview", null, null);
```

```
axExMenu1.CtlRefresh();
```

The following VFP sample adds some items that are aligned to the right:

```
With thisform.ExMenu1
```

```
.Border = 5 && BumpBorder
```

```
With .Items
```

```
With .Add("Menu 1", 2).SubMenu && EXMENUMLibCtl.SubMenu
```

```
.Add("File").Alignment = 2 && exRight
```

```
.Add("Open").Alignment = 2 && exRight
```

```
.Add ("Print Preview")
```

```
EndWith
```

```
EndWith
```

```
EndWith
```

```
thisform.ExMenu1.Object.Refresh
```


method ExMenu.Load (Stream as Variant)

Loads the menu saved using Save method.

Type	Description
Stream as Variant	A string expression that indicates the file name being loaded.

The Load method loads items from a file. The file must be saved previously using the [Save](#) method. Use the [Refresh](#) method to refresh the control's content. Use the [Images](#) property to load icons at runtime. Use the [Replacelcon](#) property to add, remove, replace, or clear the images collection.

The following VB sample loads the menu from a file:

```
With ExMenu1
    .Load "d:\temp\test.mnu"
End With
```

The following C++ sample loads the menu from a file:

```
m_menu.Load(COLEVariant( "d:\\temp\\test.mnu" ));
```

The following VB.NET sample loads the menu from a file:

```
With AxExMenu1
    .Load("d:\temp\test.mnu")
End With
```

The following C# sample loads the menu from a file:

```
axExMenu1.Load("d:\\temp\\test.mnu");
```

The following VFP sample loads the menu from a file:

```
With thisform.ExMenu1
    .Load("d:\temp\test.mnu")
EndWith
```

property ExMenu.MenuBarBorder as BorderEnum

Specifies the menu bar's border.

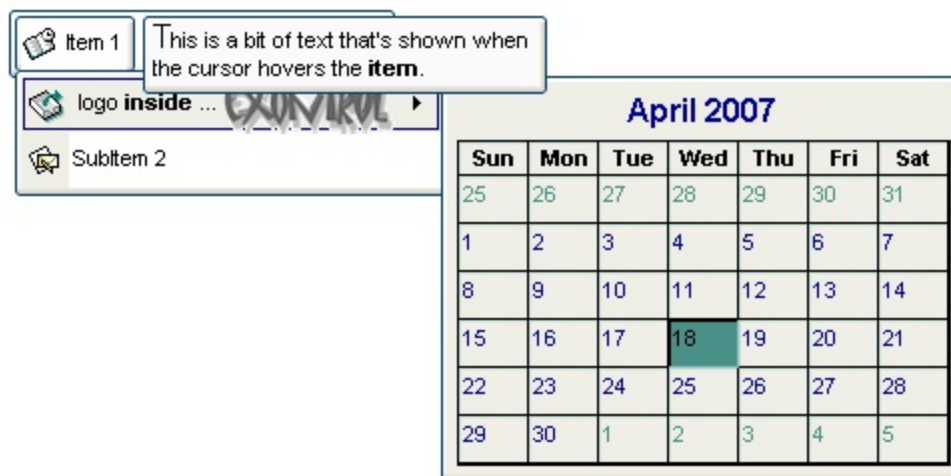
Type

Description

[BorderEnum](#)

A BorderEnum expression that defines the menu's bar border. Or a color expression whose last 7 bits in the high significant byte of the value indicates the index of the skin in the [Appearance](#) collection, being displayed as control's borders. For instance, if the Appearance = 0x1000000, indicates that the first skin object in the Appearance collection defines the control's border. **The Client object in the skin, defines the client area of the control. The items are always shown in the control's client area. The skin may contain transparent objects, and so you can define round corners. The [frame.ebn](#) file contains such of objects. Use the [eXButton's Skin builder](#) to view or change this file**

Use the MenuBarBorder property to define the border of the menu bar. The menu bar is displayed into the form. Use the [Border](#) property to define the menu's border. Use the [Appearance](#) property to define the menu's appearance. Use the [Border](#) property to define the control's borders. Use the [Font](#) property to specify the control's font. Use the [BackColor](#) property to specify the control's background color. Use the [ForeColor](#) property to specify the control's foreground color. Use the [ItemHeight](#) property to specify the height for all items. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips.



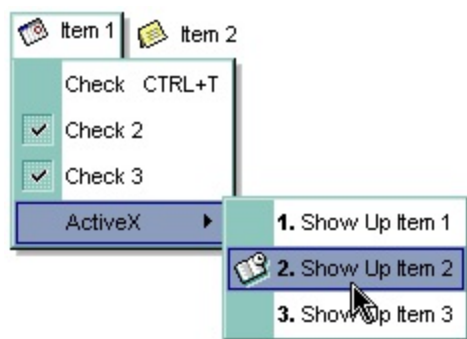
property ExMenu.OpenMode as OpenModeEnum

Specifies the way how the menu is opened.

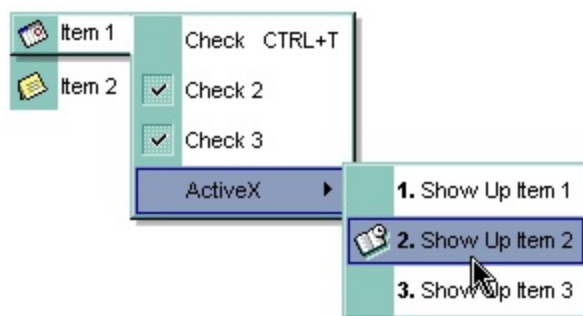
Type	Description
OpenModeEnum	An OpenModeEnum expression that indicates the way how the menu bar opens its popup menus.

By default, the OpenMode property is Vertical . If the OpenMode property is Vertical, the items in the menu bar are arranged from top to bottom, line by line, one item per line. If the OpenMode property is Horizontal the items of the menu bar are arranged from left to right. The menu bar is displayed always in the form. Use the [Appearance](#) property to specify the control's appearance. Use the [Font](#) property to specify the control's font. Use the [BackColor](#) property to specify the control's background color. Use the [ForeColor](#) property to specify the control's foreground color. The [OpenOnClick](#) property specifies whether the sub menu is opened when the user clicks the item in the menu bar.

The following screen shows shows the control when the OpenMode is Vertical:



The following screen shows shows the control when the OpenMode is Horizontal:



property ExMenu.OpenOnClick as OpenOnClickEnum

Sets or gets a value that indicates the way how the user opens the menus, using the mouse.

Type	Description
OpenOnClickEnum	An OpenOnClickEnum expression that specifies the way how the user opens the menus, using the mouse.

The OpenOnClick property specifies the way the user opens the menus. By default, the OpenOnClick property is exClickMenuBar, and so any popup menu is opened when the user clicks the menu bar . If the OpenOnClick property is exHoverMenuBar, the sub menu is opened once that the cursor hovers the item, in the menu bar. Use the [OpenMode](#) property to specify how the drop down menu is shown. The control fires the [Select](#) event when the user selects an item.

property ExMenu.Picture as IPictureDisp

Retrieves or sets the background's picture.

Type	Description
IPictureDisp	A Picture object being displayed on the control's background.

Use the Picture property to put a picture on the control's background. Use the [PictureDisplay](#) to arrange the picture on the control's background. Use the [Picture](#) property to load a picture on the drop down menu's background. Use the [BackColor](#) property to specify the control's background color. Use the [SelBackColor](#) and [SelForeColor](#) properties to define colors for selected item.



The following VB sample puts a picture on the control's background:

```
With ExMenu1
    .PictureDisplay = Tile
    .Picture = LoadPicture("c:\WinNT\Zapotec.bmp")
End With
```

The following C++ sample puts a picture on the control's background:

```
IPictureDisp* pPicture = NULL;
if ( LoadPicture( "c:\\winnt\\zapotec.bmp", &pPicture ) )
{
    m_menu.SetPicture( pPicture );
    m_menu.SetPictureDisplay( 48 /*Tile*/ );
}
```

where the LoadPicture function gets the IPictureDisp from a file:

```
#include
BOOL LoadPicture( LPCTSTR szFileName, IPictureDisp** ppPictureDisp )
```

```

{
    BOOL bResult = FALSE;
    if ( szFileName )
    {
        OFSTRUCT of;
        HANDLE hFile = NULL;;
#ifdef _UNICODE
        USES_CONVERSION;
        if ( (hFile = (HANDLE)OpenFile( W2A(szFileName), &of,, OF_READ |
OF_SHARE_COMPAT)) != (HANDLE)HFILE_ERROR )
#else
        if ( (hFile = (HANDLE)OpenFile( szFileName, &of,, OF_READ | OF_SHARE_COMPAT)) !=
(HANDLE)HFILE_ERROR )
#endif
        {
            *ppPictureDisp = NULL;
            DWORD dwHighWord = NULL, dwSizeLow = GetFileSize( hFile, &dwHighWord; );
            DWORD dwFileSize = dwSizeLow;
            HRESULT hResult = NULL;
            if ( HGLOBAL hGlobal = GlobalAlloc(GMEM_MOVEABLE, dwFileSize) )
                if ( void* pvData = GlobalLock( hGlobal ) )
                {
                    DWORD dwReadBytes = NULL;
                    BOOL bRead = ReadFile( hFile, pvData, dwFileSize, &dwReadBytes,, NULL );
                    GlobalUnlock( hGlobal );
                    if ( bRead )
                    {
                        CComPtr spStream;
                        _ASSERT( dwFileSize == dwReadBytes );
                        if ( SUCCEEDED( CreateStreamOnHGlobal( hGlobal, TRUE, &spStream; ) ) )
                            if ( SUCCEEDED( hResult = OleLoadPicture( spStream, 0, FALSE,
IID_IPictureDisp, (void**)ppPictureDisp ) ) )
                                bResult = TRUE;
                    }
                }
            CloseHandle( hFile );
        }
    }
}

```

```
}  
return bResult;  
}
```

The following VB.NET sample puts a picture on the control's background:

```
With AxExMenu1  
    .Picture = Image.FromFile("c:\winnt\zapotec.bmp")  
    .PictureDisplay = EXMENUlib.PictureDisplayEnum.Tile  
End With
```

The following C# sample puts a picture on the control's background:

```
axExMenu1.Picture = Image.FromFile("c:\\winnt\\zapotec.bmp");  
axExMenu1.PictureDisplay = EXMENUlib.PictureDisplayEnum.Tile;
```

The following VFP sample puts a picture on the control's background:

```
With thisform.ExMenu1  
    .PictureDisplay = 48 && Tile  
    .Picture = LoadPicture("c:\WinNT\Zapotec.bmp")  
EndWith
```

property ExMenu.PictureDisplay as PictureDisplayEnum

Retrieves or sets a value that indicates the way how the picture is displayed.

Type	Description
PictureDisplayEnum	A PictureDisplayEnum expression that indicates how the picture is arranged on the control's background.

The PictureDisplay property has no effect if the control's background has no picture assigned. Use the Picture property to put a picture on the control's background. Use the [PictureDisplay](#) property to arrange the picture on the background of a popup menu. Use the [Picture](#) property to load a picture on the drop down menu's background.



The following VB sample puts a picture on the control's background:

```
With ExMenu1
    .PictureDisplay = Tile
    .Picture = LoadPicture("c:\WinNT\Zapotec.bmp")
End With
```

The following C++ sample puts a picture on the control's background:

```
IPictureDisp* pPicture = NULL;
if ( LoadPicture( "c:\\winnt\\zapotec.bmp", &pPicture ) )
{
    m_menu.SetPicture( pPicture );
    m_menu.SetPictureDisplay( 48 /*Tile*/ );
}
```

where the LoadPicture function gets the IPictureDisp from a file:

```
#include
BOOL LoadPicture( LPCTSTR szFileName, IPictureDisp** ppPictureDisp )
{
```

```

BOOL bResult = FALSE;
if ( szFileName )
{
    OFSTRUCT of;
    HANDLE hFile = NULL;;
#ifdef _UNICODE
    USES_CONVERSION;
    if ( (hFile = (HANDLE)OpenFile( W2A(szFileName), &of,, OF_READ |
OF_SHARE_COMPAT)) != (HANDLE)HFILE_ERROR )
#else
    if ( (hFile = (HANDLE)OpenFile( szFileName, &of,, OF_READ | OF_SHARE_COMPAT)) !=
(HANDLE)HFILE_ERROR )
#endif
    {
        *ppPictureDisp = NULL;
        DWORD dwHighWord = NULL, dwSizeLow = GetFileSize( hFile, &dwHighWord; );
        DWORD dwFileSize = dwSizeLow;
        HRESULT hResult = NULL;
        if ( HGLOBAL hGlobal = GlobalAlloc(GMEM_MOVEABLE, dwFileSize) )
            if ( void* pvData = GlobalLock( hGlobal ) )
                {
                    DWORD dwReadBytes = NULL;
                    BOOL bRead = ReadFile( hFile, pvData, dwFileSize, &dwReadBytes,, NULL );
                    GlobalUnlock( hGlobal );
                    if ( bRead )
                        {
                            CComPtr spStream;
                            _ASSERT( dwFileSize == dwReadBytes );
                            if ( SUCCEEDED( CreateStreamOnHGlobal( hGlobal, TRUE, &spStream; ) ) )
                                if ( SUCCEEDED( hResult = OleLoadPicture( spStream, 0, FALSE,
IID_IPictureDisp, (void**)ppPictureDisp ) ) )
                                    bResult = TRUE;
                        }
                }
            CloseHandle( hFile );
    }
}

```

```
return bResult;  
}
```

The following VB.NET sample puts a picture on the control's background:

```
With AxExMenu1  
    .Picture = Image.FromFile("c:\winnt\zapotec.bmp")  
    .PictureDisplay = EXMENULib.PictureDisplayEnum.Tile  
End With
```

The following C# sample puts a picture on the control's background:

```
axExMenu1.Picture = Image.FromFile("c:\\winnt\\zapotec.bmp");  
axExMenu1.PictureDisplay = EXMENULib.PictureDisplayEnum.Tile;
```

The following VFP sample puts a picture on the control's background:

```
With thisform.ExMenu1  
    .PictureDisplay = 48 && Tile  
    .Picture = LoadPicture("c:\WinNT\Zapotec.bmp")  
EndWith
```

property ExMenu.PopupBackColor as Color

Retrieves or sets a color that indicates the popup menu's background color.

Type	Description
Color	A color expression that indicates the popup menu's background color.

Use the [ForeColor](#) and [BackColor](#) properties to customize the menu's colors. Use the [PopupForeColor](#) property to change the foreground color for the popup menu. Use the [PopupBackColor](#) property to change the background color for the popup menu. Use the [UseBackColor](#), [BackColor](#), [UseForeColor](#), [ForeColor](#) properties to define colors for any menu item. Use the [SelBackColor](#) and [SelForeColor](#) properties to define colors for selected item. Use the [Picture](#) property to load a picture on the submenu's background.

property ExMenu.PopupForeColor as Color

Retrieves or sets a color that indicates the popup menu's text color.

Type	Description
Color	A color expression that indicates the popup menu's text color.

Use the [ForeColor](#) and [BackColor](#) properties to customize the menu's colors. Use the [PopupForeColor](#) property to change the foreground color for the popup menu. Use the [PopupBackColor](#) property to change the background color for the popup menu. Use the [UseBackColor](#), [BackColor](#), [UseForeColor](#), [ForeColor](#) properties to define colors for any menu item. Use the [SelBackColor](#) and [SelForeColor](#) properties to define colors for selected item. Use the [Picture](#) property to load a picture on the submenu's background.

method ExMenu.Refresh ()

Refreshes the control.

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

Use the Refresh method if you add, remove or change items that are contained by the menu bar. The menu bar is always displayed by the control's form. Calling the Refresh method is not mandatory for changing items that are not contained directly by the menu bar.

The following VB sample calls the Refresh method:

```
ExMenu1.Refresh
```

The following C++ sample calls the Refresh method:

```
m_menu.Refresh();
```

The following VB.NET sample calls the Refresh method:

```
AxExMenu1.CtlRefresh()
```

In VB.NET the System.Windows.Forms.Control class has already a Refresh method, so the CtlRefresh method should be called.

The following C# sample calls the Refresh method:

```
axExMenu1.CtlRefresh();
```

In C# the System.Windows.Forms.Control class has already a Refresh method, so the CtlRefresh method should be called.

The following VFP sample calls the Refresh method:

```
thisform.ExMenu1.Object.Refresh()
```

method `ExMenu.RemoveAccelerator (ID as Long)`

Removes the menu item's accelerator key.

Type	Description
ID as Long	A long expression that specifies the item's identifier whom accelerator is removed.

Use the `RemoveAccelerator` method to remove an accelerator key added using the [AddAccelerator](#) method. Use the [ClearAccelerators](#) method to remove all accelerators. Use the [Remove](#) method to remove an item. Use the [Visible](#) property to hide an item. Use the [Enabled](#) property to disable an item. The item's accelerator is disabled if the item is disabled or it is hidden.

method ExMenu.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	<p>A Variant expression that specifies the icon to add or insert, as one of the following options:</p> <ul style="list-style-type: none">• a long expression that specifies the handle of the icon (HICON)• a string expression that indicates the path to the picture file• a string expression that defines the picture's content encoded as BASE64 strings using the eXImages tool• a Picture reference, which is an object that holds image data. It is often used in controls like PictureBox, Image, or in custom controls (e.g., IPicture, IPictureDisp) <p>If the Icon parameter is 0, it specifies that the icon at the given Index is removed. Furthermore, setting the Index parameter to -1 removes all icons.</p> <p>By default, if the Icon parameter is not specified or is missing, a value of 0 is used.</p>
Index as Variant	<p>A long expression that defines the index of the icon to insert or remove, as follows:</p> <ul style="list-style-type: none">• A zero or positive value specifies the index of the icon to insert (when Icon is non-zero) or to remove (when the Icon parameter is zero)• A negative value clears all icons when the Icon parameter is zero <p>By default, if the Index parameter is not specified or is missing, a value of -1 is used.</p>
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 menu control.

The following sample shows how to add a new icon to control's images list:

```
i = ExMenu1.Replacelcon( LoadPicture("d:\icons\help.ico").Handle),
```

 where `i` is the index to insert the icon

The following sample shows how to replace an icon into control's images list::

```
i = ExMenu1.Replacelcon( LoadPicture("d:\icons\help.ico").Handle, 0),
```

 in this case the `i` is zero, because the first icon was replaced.

The following sample shows how to remove an icon from control's images list:

```
ExMenu1.Replacelcon 0, i,
```

 in this case the `i` must be the index of the icon that follows to be removed

The following sample shows how to clear the control's icons collection:

```
ExMenu1.Replacelcon 0, -1
```

method ExMenu.Save (Stream as Variant)

Saves the menu to the destination.

Type	Description
Stream as Variant	A string expression that indicates the name of the file being saved.

Use the Save method to save the items of the menu to a file. Use the [Load](#) method to load items from a file. The Save method saves the items and the icons too. Use the [Images](#) property to load icons at runtime. Use the [Replacelcon](#) property to add, remove, replace, or clear the images collection.

The following VB sample saves the menu from a file:

```
With ExMenu1
    .Save "d:\temp\test.mnu"
End With
```

The following C++ sample saves the menu from a file:

```
m_menu.Save(COLEVariant( "d:\\temp\\test.mnu" ));
```

The following VB.NET sample saves the menu from a file:

```
With AxExMenu1
    .Save("d:\temp\test.mnu")
End With
```

The following C# sample saves the menu from a file:

```
axExMenu1.Save("d:\\temp\test.mnu");
```

The following VFP sample saves the menu from a file:

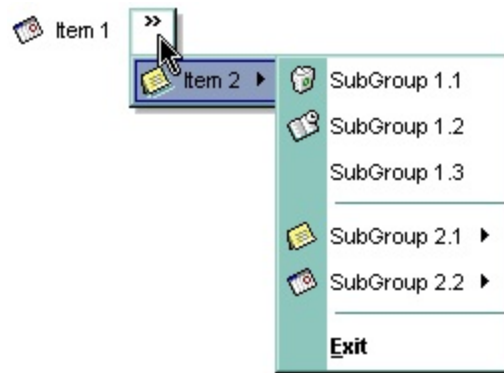
```
With thisform.ExMenu1
    .Save("d:\temp\test.mnu")
EndWith
```

property ExMenu.ScrollImage(Left as Boolean) as Long

Retrieves or sets a value that indicates the index of icon being displayed when the left or right scrolling item is visible on the menu bar.

Type	Description
Left as Boolean	A boolean expression that indicates whether the left scrolling icon is changed.
Long	A long expression that indicates the index of scrolling icon being changed.

Use the ScrollImage property to assign custom images for scrolling items. The scrolling items show on the menu bar only if they are required, and the [AllowChevron](#) property is exScroll. Use the [Images](#), [Replacelcon](#) methods to assign icons to the control.



property ExMenu.ScrollOnDrop 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 indicates the effect to be applied when the popup menu is shown. The effect is applied when user clicks an item in the menu, that has associated a sub-menu or a popup menu.

By default, the ScrollOnDrop property is exShowPopupLightUp. The ScrollOnDrop property on exShowPopupScroll specifies whether the control scrolls smoothly the drop down menu when the user clicks an item that contains a popup menu. Use the [SubMenu](#) property to access the sub menu. The control fires the [Select](#) event when the user selects an item. The control fires the [OpenPopup](#) event when the user opens a popup menu. The control fires the [ClosePopup](#) event when the user closes the popup menu. The [OpenOnClick](#) property specifies whether the control opens the sub menu when user clicks the menu bar.

property ExMenu.ScrollOnWheel as ScrollOnWheelEnum

Indicates whether the menu gets scrolled once the user rotates the mouse wheel.

Type	Description
ScrollOnWheelEnum	A ScrollOnWheelEnum expression that indicates whether the menu gets scrolled once the user rotates the mouse wheel.

By default, the ScrollOnWheel property is exScrollOnWheelJumpToArrow. The ScrollOnWheel property indicates whether the menu gets scrolled once the user rotates the mouse wheel. For instance, you can disable scrolling the menu using the mouse wheel, by setting the ScrollOnWheel property on exScrollOnWheelDisabled.

property ExMenu.SelBackColor as Color

Retrieves or sets a color that indicates the selection's background color.

Type	Description
Color	A color expression that the selection's background color. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

Use the SelBackColor and [SelfForeColor](#) properties to define colors for selected item. If the menu's [Appearance](#) property is Flat, the control draws a highlight border around the selected item. Use the [HighLightBorderColor](#) property to change the highlight border's color. Use the [BackColor](#) and [ForeColor](#) properties to customize the menu's colors. Use the [ShadowColor](#) property to specify the color used to paint the shadow border on the left side of the drop down menu, when the Appearance property is Flat. Use the [Picture](#) property to load a picture on the control's background. Use the [BackColor](#) property to specify the item's background color.

property ExMenu.SelectOn as SelectOnEnum

Specifies whether the control selects an item when user presses or releases the mouse button.

Type	Description
SelectOnEnum	A SelectOnEnum expression that indicates whether the control selects an item when user presses or releases the mouse button.

Use the SelectOn property to specify when the control selects an item. By default, the SelectOn property is exMouseDown. If the SelectOn property is exMouseUp the control selects an item only when the user releases the mouse button. Use the SelectOn property on exMouseUp, when controls under the drop down portion of the menu fire MouseUp events. The control fires the [Select](#) event when an item is selected.

property ExMenu.SelForeColor as Color

Retrieves or sets a color that indicates the selection's text color.

Type	Description
Color	A color expression that indicates the foreground color for the selected item.

Use the [SelBackColor](#) and [SelForeColor](#) properties to define colors for selected item. If the menu's [Appearance](#) property is Flat, the control draws a highlight border around the selected item. Use the [HighLightBorderColor](#) property to change the highlight border's color. Use the [BackColor](#) and [ForeColor](#) properties to customize the menu's colors. Use the [ShadowColor](#) property to specify the color used to paint the shadow border on the left side of the drop down menu, when the Appearance property is Flat. Use the [Picture](#) property to load a picture on the control's background. Use the [ForeColor](#) property to specify the item's foreground color. Use the [SelectOn](#) property to specify whether the control selects an item if the user presses or releases the mouse button.

property ExMenu.SepAcc as String

Specifies a string expression that splits the caption and accelerator key in the item.

Type	Description
String	A string expression that indicates the string being used to delimitate the caption and accelerator in the item's caption.

By default, the SepAcc property is ' ' (5 space characters). If the item's [Caption](#) contains a string defined by the SepAcc property it delimits the left side of the caption, and the right side of the caption. For instance, if you need to display the File caption on the left side of the item, and CTRL+F in the right side of the item, you need to declare an item's caption like: "File CTRL+F". As you can see there are about 5 spaces between 'File' and 'CTRL+F' strings. When the item is displayed, the File caption will be aligned to the left side of the item, since the 'CTRL+F' string will be aligned to right side of the item. Use the [Picture](#) property to load a picture on the popup menu.

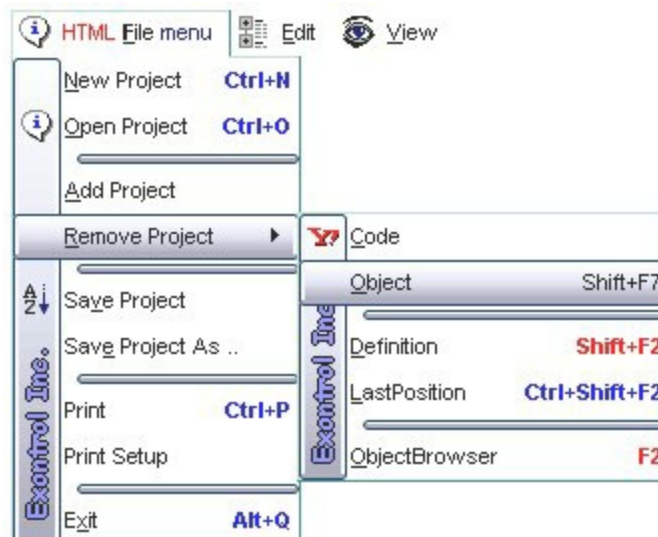


property ExMenu.ShadowColor as Color

Specifies the shadow color.

Type	Description
Color	A color expression that defines the shadow 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.

The control draws the left border fro the drop down menu, when the menu's [Appearance](#) property if Flat. Use the [ItemHeight](#) property to specify the height for all items. Use the [SelBackColor](#) and [SelForeColor](#) properties to define colors for selected item. The control fires the [Select](#) event when the user selects an item. The control displays a highlight border around the selected item ([HighLightBorderColor](#) property), when the menu's Appearance property if Flat. Use the [Border](#) property to define the control's borders. Use the [Font](#) property to specify the control's font. Use the [BackColor](#) property to specify the control's background color. Use the [ForeColor](#) property to specify the control's foreground color.



property ExMenu.ShowPopup (ID as Variant, [X as Variant], [Y as Variant], [Options as Variant]) as Long

Shows a popup menu.

Type	Description
ID as Variant	A long expression that specifies the identifier of the item whose popup will be displayed
X as Variant	A long expression that indicates the x-position to show the popup
Y as Variant	A long expression that indicates the y-position to show the popup
Options as Variant	Reservec
Long	A long expression that specifies the identifier of the item being selected

The ShowPopup method shows the popup menu of giving identifier.

property ExMenu.Template as String

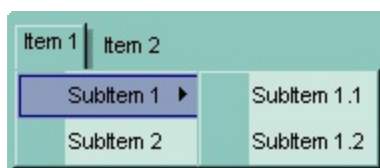
Specifies the control's template.

Type	Description
String	A string expression that indicates the list of x-script instructions.

The control's template uses the X-Script language to initialize the control's content. For instance, you can see the Template property page of the control to update the control's Template property, in design mode. Use the Template property to execute code by passing instructions as a string (template string). Use the [ToString](#) method to quick load items from a formatted string. The [ExecuteTemplate](#) property gets the result after executing a template script. The [TemplateResult](#) property returns the result of the last Template call.

For instance, the following Template adds some items to the control:

```
Items
{
  Add("Item 1",2).SubMenu
  {
    Add("SubItem 1",2).SubMenu
    {
      Add("SubItem 1.1")
      Add("SubItem 1.2")
    }
    Add("SubItem 2")
  }
  Add("Item 2",2).SubMenu
  {
    Add("SubItem 1")
    Add("SubItem 2")
  }
}
Refresh
```



At runtime, you can build this string, including CRLF sequences, and you can pass it to the Template property.

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).

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 ExMenu.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, XBasic from AlphaFive, Wonderware**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters) = Value*, so our controls provide an alternative using the TemplateDef method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be [Template](#) or [ExecuteTemplate](#) property which can use the variable a and b being defined previously.

So, calling the TemplateDef property should be as follows:

```
with (Control)
  TemplateDef = [Dim var_Column]
  TemplateDef = var_Column
  Template = [var_Column.Def(4) = 255]
endwith
```

This sample allocates a variable var_Column, assigns the value to the variable (the second call of the TemplateDef), and the Template call uses the var_Column variable (as an object), to call its Def property with the parameter 4.

Let's say we need to define the background color for a specified column, so we need to call the Def(exCellBackColor) property of the column, to define the color for all cells in the column.

The following **VB6** sample shows setting the Def property such as:

```
With Control
  .Columns.Add("Column 1").Def(exCellBackColor) = 255
  .Columns.Add "Column 2"
  .Items.AddItem 0
  .Items.AddItem 1
```

.Items.AddItem 2

End With

In **dBASE Plus**, calling the Def(4) has no effect, instead using the TemplateDef helps you to use properly the Def property as follows:

```
local Control,var_Column
```

```
Control = form.ActiveX1.nativeObject
```

```
// Control.Columns.Add("Column 1").Def(4) = 255
```

```
var_Column = Control.Columns.Add("Column 1")
```

```
with (Control)
```

```
    TemplateDef = [Dim var_Column]
```

```
    TemplateDef = var_Column
```

```
    Template = [var_Column.Def(4) = 255]
```

```
endwith
```

```
Control.Columns.Add("Column 2")
```

```
Control.Items.AddItem(0)
```

```
Control.Items.AddItem(1)
```

```
Control.Items.AddItem(2)
```

The equivalent sample for **XBasic in A5**, is as follows:

```
Dim Control as P
```

```
Dim var_Column as P
```

```
Control = topparent:CONTROL_ACTIVEX1.activex
```

```
' Control.Columns.Add("Column 1").Def(4) = 255
```

```
var_Column = Control.Columns.Add("Column 1")
```

```
Control.TemplateDef = "Dim var_Column"
```

```
Control.TemplateDef = var_Column
```

```
Control.Template = "var_Column.Def(4) = 255"
```

```
Control.Columns.Add("Column 2")
```

```
Control.Items.AddItem(0)
```

```
Control.Items.AddItem(1)
```

```
Control.Items.AddItem(2)
```

The samples just call the `Column.Def(4) = Value`, using the `TemplateDef`. The first call of `TemplateDef` property is `"Dim var_Column"`, which indicates that the next call of the `TemplateDef` will defines the value of the variable `var_Column`, in other words, it defines the object `var_Column`. The last call of the `Template` property uses the `var_Column` member to use the x-script and so to set the `Def` property so a new color is being assigned to the column.

The `TemplateDef`, [Template](#) and [ExecuteTemplate](#) support x-script language (`Template` script of the `Exontrols`), like explained below:

The `Template` or x-script is composed by lines of instructions. Instructions are separated by `"\n\r"` (newline characters) or `";"` character. The `;` character may be available only for newer versions of the components.

An x-script instruction/line can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: `Dim h, h1, h2`)*
- `variable = property(list of arguments)` *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name of the object in the context. The "list or arguments" may include variables or values separated by commas. (Sample: `h = InsertItem(0,"New Child")`)*
- `property(list of arguments) = value` *Changes the property. The value can be a variable, a string, a number, a boolean value or a RGB value.*
- `method(list of arguments)` *Invokes the method. The "list or arguments" may include variables or values separated by commas.*
- `{` *Beginning the object's context. The properties or methods called between `{` and `}` are related to the last object returned by the property prior to `{` declaration.*
- `}` *Ending the object's context*
- `object.property(list of arguments).property(list of arguments)....` *The `.` (dot) character splits the object from its property. For instance, the `Columns.Add("Column1").HeaderBackColor = RGB(255,0,0)`, adds a new column and changes the column's header back color.*

The x-script may uses constant expressions as follow:

- *boolean* expression with possible values as `True` or `False`
- *numeric* expression may starts with `0x` which indicates a hexa decimal representation, else it should starts with digit, or `+/-` followed by a digit, and `.` is the decimal separator. *Sample: `13` indicates the integer 13, or `12.45` indicates the double expression 12,45*
- *date* expression is delimited by `#` character in the format `#mm/dd/yyyy hh:mm:ss#`. *Sample: `#31/12/1971#` indicates the December 31, 1971*
- *string* expression is delimited by `"` or ``` characters. If using the ``` character, please

make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

Also , the template or x-script code may support general functions as follows:

- **Me** *property indicates the original object.*
- **RGB(R,G,B)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: BackColor = RGB(255,0,0)*
- **LoadPicture(file)** *property loads a picture from a file or from BASE64 encoded strings, and returns a Picture object required by the picture properties.*
- **CreateObject(progID)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier.*

property ExMenu.TemplateResult as Variant

Gets the result of the last Template call.

Type	Description
Variant	A VARIANT expression that indicates the result of the last Template call. The TemplateResultN property gets the result as number (double expression). The TemplateResultS property gets the result as string.

The TemplateResult, [TemplateResultN](#), [TemplateResultS](#) property returns the result of the last [Template](#) call, as variant, numeric (double) or as string. The Template property takes a string called x-script, and executes it. For instance, you can use the [TemplateDef](#), [Template](#), [TemplateResult](#) or [ExecuteTemplate](#) to work with x-script. It is known that programming languages such as **dBASE Plus**, **XBasic from AlphaFive**, **Wonderware**, 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.

For instance, the Wonderware does not support parameters for events, or parameters of any event are not defined during the event, so in this case, you require an alternative in order to get the value for these parameters. Let's say the [Select](#) event, which has one parameter ID of long type, which indicates the identifier of the item being selected. The [EventParam](#) property gets the value for any parameter of a specified event. The same, the EventParam requires parameters so Wonderware won't support it, in this case, the [Template](#) and TemplateResult can be used to get the ID parameter of the Select event as follows:

```
DIM id As Message
#exMenu1.Template = "EventParam(0)";
id = #exMenu1.TemplateResultS;
MessageBox(id, "Identifier", 0);
```

This code must be called during the Select event, else the EventParam has no effect.

The Template script (x-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 ExMenu.TemplateResultN as Double

Gets the result of the last Template call, as double.

Type	Description
Double	A Double expression that indicates the result of the last Template call. The TemplateResult property gets the result as variant. The TemplateResultS property gets the result as string.

The [TemplateResult](#), TemplateResultN, [TemplateResultS](#) property returns the result of the last [Template](#) call, as variant, numeric (double) or as string. The Template property takes a string called x-script, and executes it. For instance, you can use the [TemplateDef](#), [Template](#), [TemplateResult](#) or [ExecuteTemplate](#) to work with x-script. It is known that programming languages such as **dBASE Plus**, **XBasic from AlphaFive**, **Wonderware**, 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.

For instance, the Wonderware does not support parameters for events, or parameters of any event are not defined during the event, so in this case, you require an alternative in order to get the value for these parameters. Let's say the [Select](#) event, which has one parameter ID of long type, which indicates the identifier of the item being selected. The [EventParam](#) property gets the value for any parameter of a specified event. The same, the EventParam requires parameters so Wonderware won't support it, in this case, the [Template](#) and TemplateResult can be used to get the ID parameter of the Select event as follows:

```
DIM id As Message
#exMenu1.Template = "EventParam(0)";
id = #exMenu1.TemplateResultS;
MessageBox(id, "Identifier", 0);
```

This code must be called during the Select event, else the EventParam has no effect.

The Template script (x-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 ExMenu.TemplateResultS as String

Gets the result of the last Template call, as string.

Type	Description
String	A String expression that indicates the result of the last Template call. The TemplateResultN property gets the result as number (double expression). The TemplateResult property gets the result as variant.

The [TemplateResult](#), [TemplateResultN](#), [TemplateResultS](#) property returns the result of the last [Template](#) call, as variant, numeric (double) or as string. The [Template](#) property takes a string called x-script, and executes it. For instance, you can use the [TemplateDef](#), [Template](#), [TemplateResult](#) or [ExecuteTemplate](#) to work with x-script. It is known that programming languages such as **dBASE Plus**, **XBasic from AlphaFive**, **Wonderware**, 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.

For instance, the Wonderware does not support parameters for events, or parameters of any event are not defined during the event, so in this case, you require an alternative in order to get the value for these parameters. Let's say the [Select](#) event, which has one parameter ID of long type, which indicates the identifier of the item being selected. The [EventParam](#) property gets the value for any parameter of a specified event. The same, the [EventParam](#) requires parameters so Wonderware won't support it, in this case, the [Template](#) and [TemplateResult](#) can be used to get the ID parameter of the [Select](#) event as follows:

```
DIM id As Message
#exMenu1.Template = "EventParam(0)";
id = #exMenu1.TemplateResultS;
MessageBox(id, "Identifier", 0);
```

This code must be called during the Select event, else the EventParam has no effect.

The [Template](#) script (x-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:

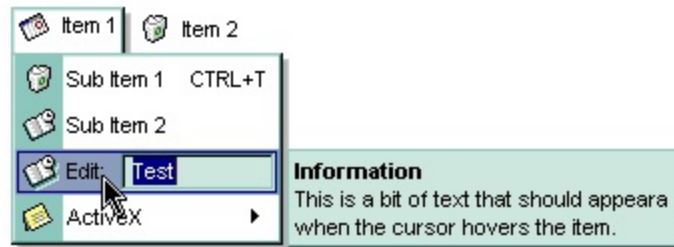
- **RGB(R,G,B)** *property retrieves an RGB value, where the R, G, B are byte values that indicates the R G B values for the color being specified. For instance, the following code changes the control's background color to red: `BackColor = RGB(255,0,0)`*
- **CreateObject(progID)** *property creates and retrieves a single uninitialized object of the class associated with a specified program identifier*

property ExMenu.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 maximum size for the tooltip's window. Use the [ToolTip](#) property to assign a tooltip to an item. Use the [ToolTipTitle](#) to assign a title to the tooltip's item.



property ExMenu.ToolTipFont as IFontDisp

Retrieves or sets the tooltip's font.

Type	Description
IFontDisp	A Font object being used to display the tooltip.

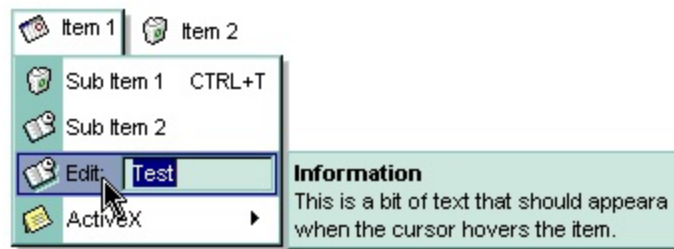
Use the ToolTipFont property to assign a font for the control's tooltip. The [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipWidth](#) property to specify the width of the tooltip window. Use the [ToolTip](#) property to assign a tooltip to an item. 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.

property ExMenu.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. Use the [ToolTipDelay](#) property specifies the time in ms that passes before the ToolTip appears. Use the [ToolTip](#) property to assign a tooltip to an item. Use the [ToolTipTitle](#) to assign a title to the tooltip's item. 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.

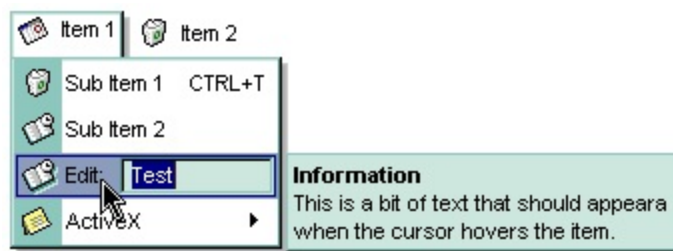


property ExMenu.ToolTipWidth as Long

Specifies a value that indicates the width of the tooltip window, in pixels.

Type	Description
Long	A long expression that indicates the width of the tooltip window, in pixels.

Use the ToolTipWidth property to specify the width of the tooltip window. The height of the tooltip window is computed based on the tooltip's description. 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 [ToolTip](#) property to assign a tooltip to an item. 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.



property ExMenu.Version as String

Retrieves the control's version.

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

The Version property defines the control's version.

property ExMenu.Visibility 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.

The Visibility property applies transparency to all sub menus. The property is supported on Windows 2000 and Windows XP. It is not supported on Windows 95, 98 or Me systems. Use the [Visible](#) property to hide an item.

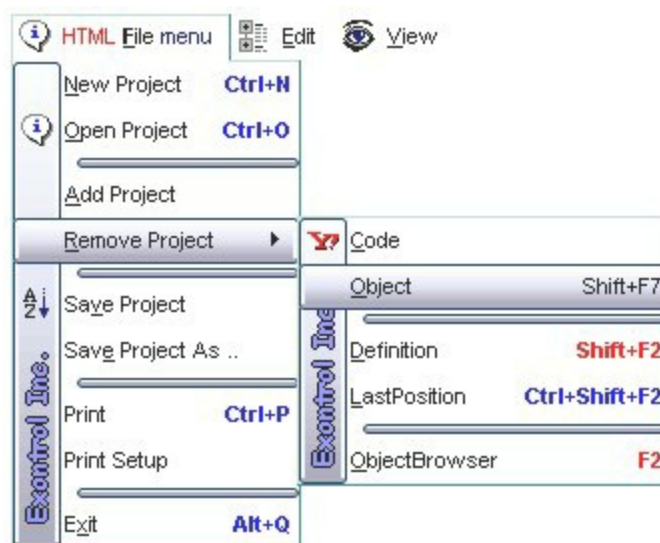


property ExMenu.VisualAppearance as Appearance

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 its simplest form, uses a single graphic file (*.ebn) assigned to a part of the control. By using a collection of objects laid over the graphic, it is possible to define which sections of the graphic will be used as borders, corners and other possible elements, fixing them to their proper position regardless of the size of the part.



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

- selected item, [SelBackColor](#) property
- shadow area, [ShadowColor](#) property
- item, [BackColor](#) property
- separator item, [Background](#) property
- and so on.

Item object

The Item object holds information about a menu item. Use the [Items](#) property to access the control's items collection. Use [Add](#) method to add new items to the menu. Use the [Item](#) or [Find](#) property to access an Item object. Here's the list of supported 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 item's background color when the UseBackColor property is True.
Bold	Specifies whether the item's caption should appear in bold.
Bullet	Retrieves or sets a value indicating whether the item is of bullet type.
Caption	Retrieves or sets a value that indicates the item's caption.
Check	Retrieves or sets a value that indicates whether the item is of check type.
CheckEffect	Specifies the visual effect of the check box in the item.
Control	Specifies whether the item contains a control inside.
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.
EditWidth	Specifies the width for the inside edit control.
Enabled	Retrieves or sets a value that indicates the item's state.
ForeColor	Specifies the item's foreground color when the UseForeColor property is True.
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.

[Parent](#)

Gets the item's parent.

[Popup](#)

Retrieves or sets a value indicating whether the item contains a sub menu.

[Separator](#)

Retrieves or sets a value that indicates whether the source is a separator item.

[ShowDown](#)

Retrieves or sets a value that indicates whether the item's submenu is up or down oriented .

[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 a Menu object that indicates the item's sub menu. Retrieves Nothing, if the item contains no sub menu.

[Tooltip](#)

Specifies the item's tooltip.

[TooltipTitle](#)

Specifies the title of the item's tooltip.

[Underline](#)

Specifies whether the item's caption appears as underlined.

[UseBackColor](#)

Retrieves or sets a value that indicates whether the item's background color is specified by BackColor property.

[UseForeColor](#)

Retrieves or sets a value that indicates whether the item's foreground color is specified by ForeColor property.

[UserData](#)

Associates an extra data to the object.

[Visible](#)

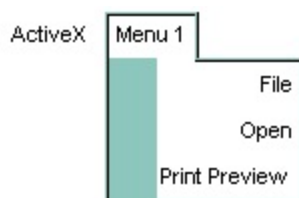
Specifies whether the item is visible or hidden.

property Item.Alignment as AlignmentEnum

Retrieves or sets the item's caption alignment.

Type	Description
AlignmentEnum	An AlignmentEnum expression that indicates the alignment of the item's caption.

Use the Alignment property to align the item's caption. By default, the item's caption is aligned to the left side of the item. Use the [Caption](#) property to specify the item's caption. Use the <r> built-in HTML tag to align parts of text to the right.



The following VB sample adds some items that are aligned to the right:

```
With ExMenu1
    .Border = BumpBorder
    With .Items
        With .Add("Menu 1", EXMENULibCtl.SubMenu).SubMenu
            .Add("File").Alignment = exRight
            .Add("Open").Alignment = exRight
            .Add ("Print Preview")
        End With
    End With
    .Refresh
End With
```

The following C++ sample adds some items that are aligned to the right:

```
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
m_menu.SetBorder( 5/*BumpBorder*/ );
CItem item = m_menu.GetItems().Add( "Menu 1", COleVariant( (long)2 /*SubMenu*/ ),
vtMissing );
CItem itemF = item.GetSubMenu().Add( "File", vtMissing, vtMissing );
itemF.SetAlignment( 2 /*RightAlignment*/ );
CItem itemO = item.GetSubMenu().Add( "Open", vtMissing, vtMissing );
```

```
itemO.SetAlignment( 2 /*RightAlignment*/ );  
item.GetSubMenu().Add("Print Preview", vtMissing, vtMissing);  
m_menu.Refresh();
```

The following VB.NET sample adds some items that are aligned to the right:

```
With AxExMenu1  
    .Border = EXMENUMLib.BorderEnum.BumpBorder  
    With .Items  
        With .Add("Menu 1", EXMENUMLib.ItemTypeEnum.SubMenu).SubMenu  
            .Add("File").Alignment = EXMENUMLib.AlignmentEnum.exRight  
            .Add("Open").Alignment = EXMENUMLib.AlignmentEnum.exRight  
            .Add("Print Preview")  
        End With  
    End With  
    .CtlRefresh()  
End With
```

The following C# sample adds some items that are aligned to the right:

```
axExMenu1.Border = EXMENUMLib.BorderEnum.BumpBorder;  
EXMENUMLib.Menu items = axExMenu1.Items;  
EXMENUMLib.Menu subMenu = items.Add("Menu 1", EXMENUMLib.ItemTypeEnum.SubMenu,  
null).SubMenu;  
subMenu.Add("File", null, null).Alignment = EXMENUMLib.AlignmentEnum.exRight;  
subMenu.Add("Open", null, null).Alignment = EXMENUMLib.AlignmentEnum.exRight;  
subMenu.Add("Print Preview", null, null);  
axExMenu1.CtlRefresh();
```

The following VFP sample adds some items that are aligned to the right:

```
With thisform.ExMenu1  
    .Border = 5 && BumpBorder  
    With .Items  
        With .Add("Menu 1", 2).SubMenu && EXMENUMLibCtl.SubMenu  
            .Add("File").Alignment = 2 && exRight  
            .Add("Open").Alignment = 2 && exRight  
            .Add ("Print Preview")  
        EndWith  
    EndWith
```

EndWith

EndWith

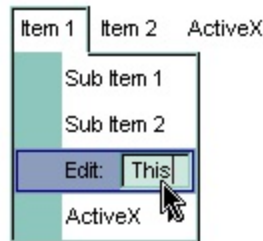
thisform.ExMenu1.Object.Refresh

property Item.AllowEdit as Boolean

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

Type	Description
Boolean	A boolean expression indicating whether the item contains an edit control.

The ExMenu ActiveX Control support edit control for any menu item. Use the [EditCaption](#) property to specify the caption for the menu item's edit. The control fires the [EditChange](#) event once that EditCaption is altered. Use the [Caption](#) property to specify the caption of the item. Use the [Enabled](#) property to disable an item. Use the [ToolTip](#) property to assign a tooltip to an item. Use the [EditWidth](#) property to specify the width of the inside edit control. Use the [EditBorder](#) property to specify the border around the edit control inside the item.



property Item.BackColor as Color

Specifies the item's background color when the UseBackColor property is True.

Type	Description
Color	A color expression that indicates the menu item's color, while UseBackColor property is True. 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 BackColor and [ForeColor](#) properties to customize the colors for an menu item. If the [UseBackColor](#) property is False, the BackColor property has no effect. The BackColor property is applied to the menu item only if the [UseBackColor](#) property is True. Use the <bgcolor> HTML tag in the [Caption](#) property to specify the background color for parts of the caption. Changing the BackColor property automatically sets the UseBackColor property on True. Use the [PopupBackColor](#) property to change the background color for the popup menu.



The following VB sample changes the background color for the item with the identifier 50:

```
With ExMenu1.Item(50)  
    .BackColor = vbRed  
End With
```

The following C++ sample changes the background color for the item with the identifier 50:

```
CItem item = m_menu.GetItem( COleVariant( long( 50 ) ) );  
item.SetBackColor( RGB(255,0,0) );
```

The following VB.NET sample changes the background color for the item with the identifier 50:

```
With AxExMenu1.item(50)
    .BackColor = ToUInt32(Color.Red)
End With
```

where the ToUInt32 function converts a Color expression to OLE_COLOR:

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

The following C# sample changes the background color for the item with the identifier 50:

```
EXMENUMLib.item item = axExMenu1[50];
item.BackColor = ToUInt32(Color.Red);
```

where the ToUInt32 function converts a Color expression to OLE_COLOR:

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

The following VFP sample changes the background color for the item with the identifier 50:

```
With thisform.ExMenu1.Item(50)
    .BackColor = RGB(255,0,0)
EndWith
```

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 should appear in bold.

Use the Bold, [Italic](#), [Underline](#) and [Strikeout](#) properties to customize the font attribute for any menu item. Use the HTML tag in the [Caption](#) property to specify that parts of the caption should appear in bold. Use the [Font](#) property to specify the control's font.

The following VB sample bolds the item with the identifier 50:

```
With ExMenu1.Item(50)
    .Bold = True
End With
```

The following C++ sample bolds the item with the identifier 50:

```
CItem item = m_menu.GetItem( COleVariant( long( 50 ) ) );
item.SetBold( TRUE );
```

The following VB.NET sample bolds the item with the identifier 50:

```
With AxExMenu1.item(50)
    .Bold = True
End With
```

The following C# sample bolds the item with the identifier 50:

```
EXMENUlib.item item = axExMenu1[50];
item.Bold = true;
```

The following VFP sample bolds the item with the identifier 50:

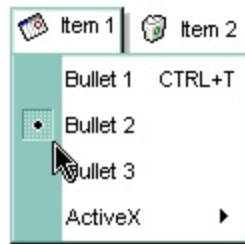
```
With thisform.ExMenu1.Item(50)
    .Bold = .t.
EndWith
```

property Item.Bullet as Boolean

Retrieves or sets a value indicating whether the item is of bullet type.

Type	Description
Boolean	A boolean expression whether the item is of bullet type.

The control displays a bullet left to the item's caption. Use the [Caption](#) property to specify the caption of the item. The bullet is displayed only if the [Check](#) property is True. The control fires the [Select](#) event when the user selects an item. Use the [Image](#) property to assign an icon to an item. Use the [CheckEffect](#) property to change the visual effect of the check box or bullet inside the item.



property Item.Caption as String

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

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

Use the Caption property to change the item's caption. Use the [EditCaption](#) property to change the caption for the edit control inside the item, if the [AllowEdit](#) property is True. Use the [Add](#) method to specify the caption of the item when adding the item. The [SepAcc](#) property splits the caption and accelerator key in the item. Use the [Bold](#) property to bold the entire item. Use the [ID](#) property to identify an item. Use the [ToolTip](#) property to assign a tooltip to an item. Use the [ForeColor](#) property to specify the item's foreground color.

The Caption property may contain built-in HTML tags like follows:

- ` bold `
- `<u> underline </u>`
- `<s> strikeouts </s>`
- `<i> italic </i>`
- `<fgcolor = FF0000> fgcolor </fgcolor>`
- `<bgcolor = FF0000> bgcolor </bgcolor>`
- `
` breaks a line.
- `<solidline>` draws a solid line
- `<dotline>` draws a dotted line
- `<upline>` draws the line to the top of the text line
- `<r>` aligns the rest of the text line to the right side.
- `number[:width]` inserts an icon inside the cell's caption. The number indicates the index of the icon being inserted. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- `key[:width]` inserts a custom size picture being loaded using the [HTMLPicture](#) property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.
- `text ` displays portions of text with a different font and/or different size. For instance, the `bit` draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, `bit` displays the bit text using the current font, but with a different size.

Newer HTML format supports subscript and superscript like follows:

- **<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

Also, newer HTML format supports decorative text like follows:

- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The HTML tag can be used to define the height of the font. Any of the 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.Check as Boolean

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

Type	Description
Boolean	A boolean expression that indicates whether the item is of check type.

Checks or uncheck the item. The control displays a check box left to the item's caption. Use the [Caption](#) property to specify the caption of the item. If the Check property is True, the control displays a check box, if the [Bullet](#) property is False, else the control displays a bullet, if the Bullet property is True. The control fires the [Select](#) event when the user selects an item. Use the [Image](#) property to assign an icon to an item. Use the [CheckEffect](#) property to change the visual effect of the check box or bullet inside the item.



The following VB sample checks or uncheck the items (30,40,50) when the user selects any of them:

```
Private Sub ExMenu1_Select(ByVal ID As Long)
    Dim bChange As Boolean
    bChange = False
    Select Case ID
        Case 30
            bChange = True
        Case 40
            bChange = True
        Case 50
            bChange = True
    End Select
    If (bChange) Then
        ExMenu1.Item(ID).Check = Not ExMenu1.Item(ID).Check
    End If
End Sub
```

The following C++ sample checks or uncheck the items (30,40,50) when the user selects any of them:

```
void OnSelectExmenu1(long ID)
{
    switch ( ID )
    {
        case 30:
        case 40:
        case 50:
        {
            CItem item = m_menu.GetItem( COleVariant( ID ) );
            item.SetCheck( !item.GetCheck() );
            break;
        }
    }
}
```

The following VB.NET sample checks or uncheck the items (30,40,50) when the user selects any of them:

```
Private Sub AxExMenu1_SelectEvent(ByVal sender As System.Object, ByVal e As
AxEXMENULib.IMenuEvents_SelectEvent) Handles AxExMenu1.SelectEvent
    Dim bChange As Boolean = False
    Select Case e.iD
        Case 30
            bChange = True
        Case 40
            bChange = True
        Case 50
            bChange = True
    End Select
    If (bChange) Then
        AxExMenu1.item(e.iD).Check = Not AxExMenu1.item(e.iD).Check
    End If
End Sub
```

The following C# sample checks or uncheck the items (30,40,50) when the user selects any of them:

```

private void axExMenu1_SelectEvent(object sender,
AxEXMENULib._IMenuEvents_SelectEvent e)
{
    switch (e.iD)
    {
        case 30:
        case 40:
        case 50:
            {
                EXMENULib.item item = axExMenu1[e.iD];
                item.Check = !item.Check;
                break;
            }
    }
}

```

The following VFP sample checks or uncheck the items (30,40,50) when the user selects any of them:

```

*** ActiveX Control Event ***
LPARAMETERS id

local bChange
bChange = .f.
do case
    case id = 30
        bChange = .t.
    case id = 40
        bChange = .t.
    case id = 50
        bChange = .t.
endcase
if ( bChange )
    with thisform.ExMenu1.Item(id)
        .Check = !.Check()
    endwith
endif

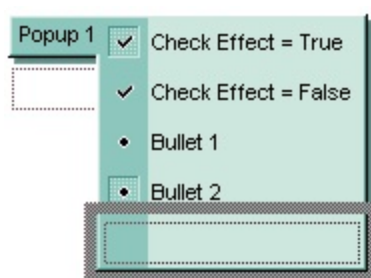
```


property Item.CheckEffect as Boolean

Specifies the visual effect of the check box in the item.

Type	Description
Boolean	A Boolean expression that specifies the visual effect of the checkbox in the item.

By default, the CheckEffect property is True. Use the CheckEffect property to remove the small rectangle around the check box. The [Check](#) property specifies whether the item displays a check box. The [Bullet](#) property specifies whether the item should display a bullet instead a check box. The CheckEffect property has effect only if the Check property is True, or Bullet is True.



property Item.Control as Boolean

Specifies whether the item contains a control inside.

Type	Description
Boolean	A boolean expression that indicates whether the item hosts an ActiveX control.

The Control property specify whether the item hosts an ActiveX control. Use the [SubControl](#) property to access the item's inside ActiveX control. The Control property retrieves True, if the ItemType parameter is SubControl when calling the [Add](#) method. Use the [Caption](#) property to specify the caption of the item. The control fires the [OleEvent](#) event when an inside ActiveX control fires an event.

property Item.Cursor as Variant

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

Type	Description
Variant	A string expression that indicates a predefined value listed below, a string expression that indicates the path to a cursor file, a long expression that indicates the handle of the cursor.

Use the Cursor property to specify the cursor that control displays when the mouse pointer hovers the item.

Here's the list of predefined values (string expressions):

- **"exDefault"** - (Default) Shape determined by the object.
- **"exArrow"** - Arrow.
- **"exCross"** - Cross (cross-hair pointer).
- **"exIBeam"** - I Beam.
- **"exIcon"** - Icon (small square within a square).
- **"exSize"** - Size (four-pointed arrow pointing north, south, east, and west).
- **"exSizeNESW"** - Size NE SW (double arrow pointing northeast and southwest).
- **"exSizeNS"** - Size N S (double arrow pointing north and south).
- **"exSizeNWSE"** - Size NW, SE.
- **"exSizeWE"** - Size W E (double arrow pointing west and east).
- **"exUpArrow"** - Up Arrow.
- **"exHourglass"** - Hourglass (wait).
- **"exNoDrop"** - No Drop.
- **"exArrowHourglass"** - Arrow and hourglass.
- **"exHelp"** - Arrow and question mark.
- **"exSizeAll"** - Size all.

If the cursor value is a string expression, the control looks first if it is not a predefined value like listed above, and if not, it tries to load the cursor from a file. If the Cursor property is a long expression it always indicates a handle to a cursor. The API functions like: LoadCursor or LoadCursorFromFile retrieves a handle to a cursor. In .NET framework, the Handle parameter of the Cursor object specifies the handle to the cursor. Use the Cursors object to access to the list of predefined cursors in the .NET framework.

The following VB sample changes the cursor while the mouse pointer hovers the item:

```
With ExMenu1(20)  
    .Cursor = "exCross"
```

```
End With
```

Here's the VB.NET alternative:

```
AxExMenu1.Ctlset_Cursor(EXEDITLib.ClientAreaEnum.exLineNumberArea, "exCross")
```

The following VB sample loads a cursor from a file:

```
With ExMenu1  
    .Cursor = "C:\WINNT\Cursors\metronom.ani"  
End With
```

And here's the VB.NET alternative:

```
AxExMenu1.Ctlset_Cursor("C:\WINNT\Cursors\metronom.ani")
```

The following VB.NET sample changes the cursor with one that Cursors object defines (PanEast cursor):

```
AxExMenu1.Ctlset_Cursor(Cursors.PanEast.Handle)
```

The following C++ sample loads the cursor from a file:

```
m_edit.SetCursor( COleVariant("C:\\WINNT\\Cursors\\metronom.ani" ) );
```

The following C# sample loads the cursor from a file:

```
axExMenu1.Ctlset_Cursor("C:\\WINNT\\Cursors\\metronom.ani");
```

The following VFP sample loads the cursor from a file:

```
with thisform.ExMenu1.Object  
    .Cursor = "C:\WINNT\Cursors\metronom.ani"  
endwith
```

property Item.EditBorder as EditBorderEnum

Specifies the border for the inside edit control.

Type	Description
EditBorderEnum	An EditBorderEnum expression that indicates the border for the edit control inside the item.

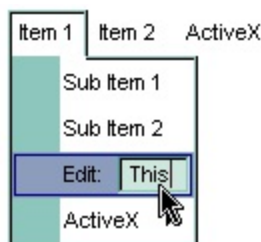
By default, the EditBorder property is exEditBorderInset. Use the EditBorder property to specify the border around the edit control inside the item. Use the [AllowEdit](#) property to specify whether the item displays an edit control. Use the [EditWidth](#) property to specify the width of the inside edit control. Use the EditBorder property to change the edit control's border.

property Item.EditCaption as String

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

Type	Description
String	A string expression that defines the caption for the item's edit.

Use the EditCaption property to get the caption of the item's edit. Use the [AllowEdit](#) property to let the menu item hosts an edit control. Use the [Caption](#) property to specify the caption of the item. Use the [Create](#) method to insert an ActiveX control. Use the [Enabled](#) property to disable an item. Use the [ToolTip](#) property to assign a tooltip to an item. Use the [EditWidth](#) property to specify the width of the inside edit control. Use the [EditBorder](#) property to specify the border around the edit control inside the item.



property Item.EditWidth as Long

Specifies the width for the inside edit control.

Type	Description
Long	A Long expression that indicates the width of the inside edit control. If the value is negative, the absolute value indicates the minimum width for the inside edit control. If the value is positive it indicates the exactly width of the inside edit control.

The EditWidth property specifies the width of the inside edit control, if the [AllowEdit](#) property is True. By default, the EditWidth property is -32, that indicates the minimum width of the inside edit control as being 32 pixels. Use the [EditBorder](#) property to specify the border around the edit control inside the item. Use the EditWidth property to specify the width of the inside edit control. The property has no effect if the AllowEdit property is False. For instance, the Item.EditWidth = 150, indicates that the inside edit control has exactly 150 pixels. If the Item.EditWidth = -150 it indicates that the width of the inside edit control can't be less than 150 pixels.

property Item.Enabled as Boolean

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

Type	Description
Boolean	A color expression that indicates the item's state.

The Enabled property enables or disables an item. Use the [Visible](#) property to hide or show an item. The user can't select a disabled item. The control fires the [Select](#) event when the user selects a new item. Use the [ForeColor](#) property to specify the item's foreground color. Use the [BackColor](#) property to specify the item's background color. The control changes the item's appearance when painting a disabled item.

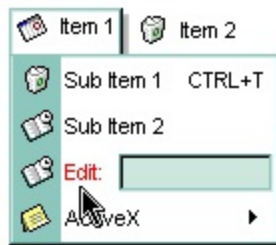


property Item.ForeColor as Color

Specifies the item's foreground color when the UseForeColor property is True.

Type	Description
Color	A color expression that specifies the item's foreground color when the UseForeColor property is True.

Use [BackColor](#) and [ForeColor](#) properties to customize the colors for an menu item. If the [UseForeColor](#) property is False, the [BackColor](#) property has no effect. The [BackColor](#) property is applied to the menu item only if the [UseForeColor](#) property is True. Use the <fgcolor> HTML tag in the [Caption](#) property to specify the background color for parts of the caption. The [UseForeColor](#) property is set on True, when the user changes the [ForeColor](#) property.



The following VB sample changes the foreground color for the item with the identifier 50:

```
With ExMenu1.Item(50)
    .ForeColor = vbRed
End With
```

The following C++ sample changes the foreground color for the item with the identifier 50:

```
CItem item = m_menu.GetItem( COleVariant( long( 50 ) ) );
item.SetForeColor( RGB(255,0,0) );
```

The following VB.NET sample changes the foreground color for the item with the identifier 50:

```
With AxExMenu1.item(50)
    .ForeColor = ToUInt32(Color.Red)
End With
```

where the ToUInt32 function converts a Color expression to OLE_COLOR:

```
Shared Function ToUInt32(ByVal c As Color) As UInt32
```

```
Dim i As Long
```

```
i = c.R
```

```
i = i + 256 * c.G
```

```
i = i + 256 * 256 * c.B
```

```
ToUInt32 = Convert.ToUInt32(i)
```

```
End Function
```

The following C# sample changes the foreground color for the item with the identifier 50:

```
EXMENUMLib.item item = axExMenu1[50];  
item.ForeColor = ToUInt32(Color.Red);
```

where the ToUInt32 function converts a Color expression to OLE_COLOR:

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

The following VFP sample changes the foreground color for the item with the identifier 50:

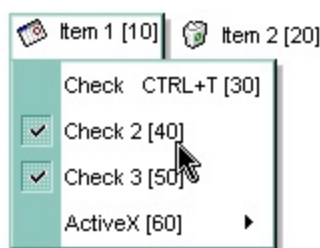
```
With thisform.ExMenu1.Item(50)  
    .ForeColor = RGB(255,0,0)  
EndWith
```

property Item.ID as Long

Retrieves or sets a value that specifies the item's identifier.

Type	Description
Long	A long expression that specifies the item's identifier.

It is recommended to use unique identifier for the menu items, because the [Find](#) or [Item](#) property finds only the first item that matches the identifier. The identifier is not used by the control in any way, so it can be taken as a extra data as well. You can assign the item's identifier when you add new items using the [Add](#) method. Use the [Debug](#) property to display the item identifiers when the control is running. Use the [Caption](#) property to specify the caption of the item.



property Item.Image as Long

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

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

Use the Image property to attach an icon to the menu item. The index of the image being used is zero based. If the Image property is negative no icon is associated. Use the [Images](#) method to assign a list of icons at runtime. Use the [Check](#) property to assign a check box to the item. Use the [Bullet](#) property to assign a radio button to an item. Use the [Caption](#) property to specify the caption of item. Use the [BackColor](#) property to specify the item's background color.

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 should appear in italic.

Use the [Bold](#), [Italic](#), [Underline](#) and [Strikeout](#) properties to customize the font attribute for any menu item. Use the HTML tag in the [Caption](#) property to specify that parts of the caption should appear in italic. Use the [Font](#) property to specify the control's font.

The following VB sample makes italic the item with the identifier 50:

```
With ExMenu1.Item(50)
    .Italic = True
End With
```

The following C++ sample makes italic the item with the identifier 50:

```
CItem item = m_menu.GetItem( COleVariant( long( 50 ) ) );
item.SetItalic( TRUE );
```

The following VB.NET sample makes italic the item with the identifier 50:

```
With AxExMenu1.item(50)
    .Italic = True
End With
```

The following C# sample makes italic the item with the identifier 50:

```
EXMENUMLib.item item = axExMenu1[50];
item.Italic = true;
```

The following VFP sample makes italic the item with the identifier 50:

```
With thisform.ExMenu1.Item(50)
    .Italic = .t.
EndWith
```

property Item.Parent as Item

Gets the item's parent.

Type	Description
Item	An Item object that identifies the parent of the item.

Use the Parent property to get the parent's item. The Parent property retrieves nothing if the item has no parent. The items in the [Items](#) collection has no parent items. Use the [Add](#) method to add new items to your menu. Use the [SubMenu](#) property to access the item's sub menu, if the item is of SubMenu type. Use the [SubControl](#) property to access the inside ActiveX control, if the item is of SubControl type.

property Item.Popup as Boolean

Retrieves or sets a value indicating whether the item contains a sub menu.

Type	Description
Boolean	A boolean value indicating whether the item contains a sub menu.

If the item contains no sub menu, the Popup property retrieves False. If the Popup retrieves True, the [SubMenu](#) property gets the item's submenu. Use the [Add](#) method to insert a sub menu. Use the Popup property to convert on the fly a regular item to a popup item. Use the [ShowDown](#) property to specify whether the popup menu should show up or down.

The following VB sample shows adds items to a menu of popup type:

```
With ExMenu1.Items
  Dim i As Item
  Set i = .Add("Popup", , 1234)
  i.Popup = True
  With i.SubMenu
    .Add("Child").Image = 1
  End With
End With
ExMenu1.Refresh
```

The following C++ sample adds items to a menu of popup type:

```
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
CItem item = m_menu.GetItems().Add( "Popup", vtMissing, COleVariant( (long)1234 ) );
item.SetPopup( TRUE );
item.GetSubMenu().Add( "Child 1", vtMissing , vtMissing ).SetImage( 1 );
m_menu.Refresh();
```

The following VB.NET sample adds items to a menu of popup type:

```
With AxExMenu1.Items
  Dim i As EXMENUlib.item = .Add("Popup", , 1234)
  i.Popup = True
  With i.SubMenu
```

```
.Add("Child").Image = 1
End With
End With
AxExMenu1.CtlRefresh()
```

The following C# sample adds items to a menu of popup type:

```
EXMENUMLib.item item = axExMenu1.Items.Add("Popup",null, 1234);
item.Popup = true;
EXMENUMLib.Menu subMenu = item.SubMenu;
subMenu.Add("Child 1", null, null).Image = 1;
axExMenu1.CtlRefresh();
```

The following VFP sample adds items to a menu of popup type:

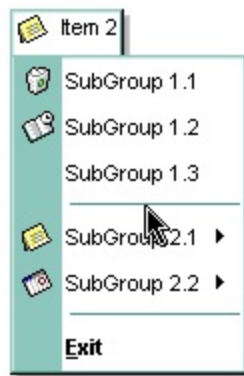
```
With thisform.ExMenu1.Items
  with .Add("Popup", , 1234)
    .Popup = .t.
    With .SubMenu
      .Add("Child").Image = 1
    EndWith
  endwith
EndWith
thisform.ExMenu1.Object.Refresh
```

property Item.Separator as Boolean

Retrieves or sets a value that indicates whether the source is a separator item.

Type	Description
Boolean	A boolean expression that indicates whether the source is a separator item.

The Separator item can't be selected. Use the [Add](#) method to insert a separator item. Use the Separator property to divide group of items in the same submenu. use the [SubMenu](#) property to access the item's sub menu. Use the [Popup](#) property to specify whether the item hosts a sub menu. Use the [Separator](#) property to make a regular item a separator item, and reverse.



The following VB sample adds a separator item to a submenu :

```
With ExMenu1.Item(10).SubMenu  
    .Add "", EXMENUMLibCtl.Separator  
End With
```

The following C++ sample adds a separator item to a submenu :

```
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;  
m_menu.GetItem( COleVariant( (long)10 ) ).GetSubMenu().Add( "", COleVariant( long(1)  
/*Separator*/ ), vtMissing );
```

The following VB.NET sample adds a separator item to a submenu :

```
With AxExMenu1.item(10).SubMenu  
    .Add("", EXMENUMLib.ItemTypeEnum.Separator)  
End With
```

The following C# sample adds a separator item to a submenu :

```
axExMenu1[10].SubMenu.Add("", EXMENULib.ItemTypeEnum.Separator, null);
```

The following VFP sample adds a separator item to a submenu :

```
With thisform.ExMenu1.Item(10).SubMenu  
  .Add("", 1) && Separator  
EndWith
```

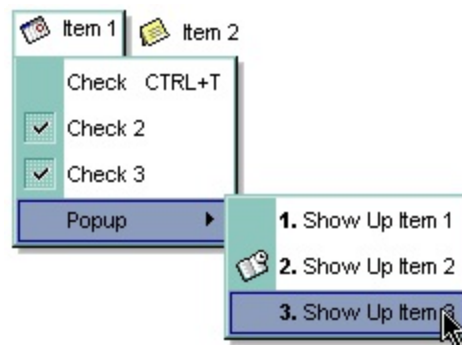
property Item.ShowDown as Boolean

Retrieves or sets a value that indicates whether the item's submenu is up or down oriented .

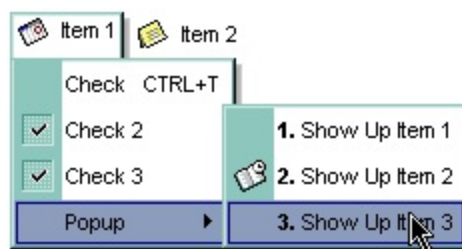
Type	Description
Boolean	A boolean expression that indicates whether the item's submenu is up or down oriented.

By default, a sub menu is shown down. Use the ShowDown property to display the sub menu up or down. Use the [Popup](#) property to specify whether an item hosts a popup menu. Use the [Separator](#) property to specify whether an item is a separator item. Use the [Add](#) method to add a popup menu, an ActiveX Item, or a separator menu.

The following screen displays the Popup's sub menu if the ShowDown property is True:



The following screen displays the Popup's sub menu if the ShowDown property is False:



The following VB sample adds a submenu that shows :

```
With ExMenu1
    .Border = BumpBorder
    With .Items
        With .Add("Menu 1", EXMENULibCtl.SubMenu)
            .ShowDown = False
            With .SubMenu
                .Add("File").Alignment = exRight
                .Add("Open").Alignment = exRight
                .Add ("Print Preview")
```

End With

End With

End With

.Refresh

End With

The following C++ sample adds some items that are aligned to the right:

```
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
m_menu.SetBorder( 5/*BumpBorder*/ );
CItem item = m_menu.GetItems().Add( "Menu 1", COleVariant( (long)2 /*SubMenu*/ ),
vtMissing );
item.SetShowDown( FALSE );
CItem itemF = item.GetSubMenu().Add( "File", vtMissing, vtMissing );
itemF.SetAlignment( 2 /*RightAlignment*/ );
CItem itemO = item.GetSubMenu().Add( "Open", vtMissing, vtMissing );
itemO.SetAlignment( 2 /*RightAlignment*/ );
item.GetSubMenu().Add("Print Preview", vtMissing, vtMissing);
m_menu.Refresh();
```

The following VB.NET sample adds some items that are aligned to the right:

```
With AxExMenu1
.Border = EXMENUMLib.BorderEnum.BumpBorder
With .Items
With .Add("Menu 1", EXMENUMLib.ItemTypeEnum.SubMenu)
.ShowDown = False
With .SubMenu
.Add("File").Alignment = EXMENUMLib.AlignmentEnum.exRight
.Add("Open").Alignment = EXMENUMLib.AlignmentEnum.exRight
.Add("Print Preview")
End With
End With
End With
.CtlRefresh()
End With
```

The following C# sample adds some items that are aligned to the right:

```
axExMenu1.Border = EXMENUMLib.BorderEnum.BumpBorder;
EXMENUMLib.Menu items = axExMenu1.Items;
EXMENUMLib.item item = items.Add("Menu 1", EXMENUMLib.ItemTypeEnum.SubMenu, null);
item.ShowDown = false;
EXMENUMLib.Menu subMenu = item.SubMenu;
subMenu.Add("File", null, null).Alignment = EXMENUMLib.AlignmentEnum.exRight;
subMenu.Add("Open", null, null).Alignment = EXMENUMLib.AlignmentEnum.exRight;
subMenu.Add("Print Preview", null, null);
axExMenu1.CtlRefresh();
```

The following VFP sample adds some items that are aligned to the right:

```
With thisform.ExMenu1
  .Border = 5 && BumpBorder
  With .Items
    With .Add("Menu 1", 2)
      .ShowDown = .f.
      with .SubMenu && EXMENUMLibCtl.SubMenu
        .Add("File").Alignment = 2 && exRight
        .Add("Open").Alignment = 2 && exRight
        .Add ("Print Preview")
      endwith
    EndWith
  EndWith
EndWith
thisform.ExMenu1.Object.Refresh
```

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 should appear in strikeout .

Use the [Bold](#), [Italic](#), [Underline](#) and **Strikeout** properties to customize the font attribute for any menu item. Use the <s> HTML tag in the [Caption](#) property to specify that parts of the caption should appear in **strikeout**. Use the [Font](#) property to specify the control's font.

The following VB sample makes **strikeout** the item with the identifier 50:

```
With ExMenu1.Item(50)
    .Strikeout = True
End With
```

The following C++ sample makes **strikeout** the item with the identifier 50:

```
CItem item = m_menu.GetItem( COleVariant( long( 50 ) ) );
item.SetStrikeout( TRUE );
```

The following VB.NET sample makes **strikeout** the item with the identifier 50:

```
With AxExMenu1.item(50)
    .Strikeout = True
End With
```

The following C# sample makes **strikeout** the item with the identifier 50:

```
EXMENUlib.item item = axExMenu1[50];
item.Strikeout = true;
```

The following VFP sample makes **strikeout** the item with the identifier 50:

```
With thisform.ExMenu1.Item(50)
    .Strikeout = .t.
EndWith
```


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 information about an inside component.

Use the SubControl property when the item holds an ActiveX control. The [Control](#) property specifies whether an item holds an ActiveX control. The [OleEvent](#) event is fired each time when an inside component fires an event. Use the [ControlID](#) property to specify the ActiveX identifier. Use the [Create](#) method to create the item's ActiveX control.

The following VB sample adds the [Exontrol.ChartView](#) component:

```
With ExMenu1.Items
  With .Add(" ActiveX ", EXMENULibCtl.ItemTypeEnum.SubControl, 1234).SubControl
    .Width = 256
    .Height = 256
    .ControlID = "Exontrol.ChartView"
    .Create
  With .Object
    .BeginUpdate
    .BackColor = RGB(255, 255, 255)
    .Appearance = 2
    .HasButtons = 3
    .ButtonsAlign = 0
    .PenWidthLink = 3
  With .Root
    .Caption = "RootSome information here."
  Line 1:1
  Line 2:2"
    .Image = 1
    .AddAssistant ("Assistant node")
  End With
  With .Nodes
    With .Add("Item 1", , "Key1")
      .HasButton = False
      .LinkTo = "Key2"
```

```
End With
```

```
.Add "SubItem 1", "Key1"
```

```
With .Add("Sub Item 2", , "Key2")
```

```
Dim s As String
```

```
s =
```

```
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCAbigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBl
```

```
s = s +
```

```
"fFSEBhikGxSDKbgnglBgoCAAQ7F6IxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE"
```

```
.Picture = s
```

```
.Expanded = False
```

```
.ArrangeSiblingNodesAs = 1
```

```
End With
```

```
.Add "SubItem 1", "Key2"
```

```
.Add "SubItem 2", "Key2"
```

```
End With
```

```
.EndUpdate
```

```
End With
```

```
End With
```

```
End With
```

```
ExMenu1.Refresh
```

The following C++ sample adds the Exontrol.ChartView component:

```
#include "Item.h"
```

```
#include "Menu.h"
```

```
#include "Control.h"
```

```
#import
```

```
CItem item = m_menu.GetItems().Add( " ActiveX ", COleVariant( (long)3 /*SubControl*/ ),  
COleVariant( (long)1234 ) );
```

```
CControl control = item.GetSubControl();
```

```
control.SetWidth( 256 );
```

```
control.SetHeight( 256 );
```

```
control.SetControlID( "Exontrol.ChartView" );
```

```
control.Create();
```

```
EXORGCHARTLib::IChartViewPtr spChart( control.GetObject() );
```

```
if ( spChart != NULL )
```

```
{
```

```
    spChart->BeginUpdate();
```

```
    spChart->BackColor = RGB(255, 255, 255);
```

```
    spChart->Appearance = EXORGCHARTLib::Sunken;
```

```
    spChart->HasButtons = EXORGCHARTLib::exWPlus;
```

```
    spChart->ButtonsAlign = EXORGCHARTLib::UpperLeft;
```

```
    spChart->PenWidthLink = 3;
```

```
    EXORGCHARTLib::INodePtr spRoot = spChart->Root;
```

```
    spRoot->Caption = "Root
```

Some information here.

```
Line 1:1
```

```
Line 2:2";
```

```
    spRoot->Image = 1;
```

```
    spRoot->AddAssistant ("Assistant node", vtMissing, vtMissing);
```

```
    EXORGCHARTLib::INodesPtr spNodes = spChart->Nodes;
```

```
    EXORGCHARTLib::INodePtr spNode1 = spNodes->Add("Item 1", vtMissing , "Key1",
```

```
vtMissing, vtMissing);
```

```
    spNode1->HasButton = false;
```

```
    spNode1->LinkTo = "Key2";
```

```
    spNodes->Add("SubItem 1", "Key1", vtMissing, vtMissing, vtMissing );
```

```
    EXORGCHARTLib::INodePtr spNode2 = spNodes->Add("Sub Item 2", vtMissing, "Key2",
```

```
vtMissing, vtMissing );
```

```
    CString s(
```

```
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCAbigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBI  
);
```

```
    s = s +
```

```
"fFSEBhikGxSDKbgnglBgoCAAQ7F6IxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE"
```

```
    spNode2->put_Picture( COleVariant( s ) );
```

```
    spNode2->Expanded = false;
```

```
    spNode2->ArrangeSiblingNodesAs = EXORGCHARTLib::exHorizontally;
```

```
    spNodes->Add("SubItem 1", "Key2", vtMissing, vtMissing, vtMissing );
```

```
    spNodes->Add("SubItem 2", "Key2", vtMissing, vtMissing, vtMissing );
```

```
    spChart->EndUpdate();
```

```
}
```

```
m_menu.Refresh();
```

The C++ sample requires calling the `#import <exorgchart.dll>` to import definitions for the `Exontrol.ChartView` component. It generates the `EXORGCHARTLib` namespace where you can find all objects of the `ExOrgChart` component.

The following VB.NET sample adds the `Exontrol.ChartView` component:

```
With AxExMenu1.Items
```

```
  With .Add(" ActiveX ", EXMENULib.ItemTypeEnum.SubControl, 1234).SubControl
```

```
    .Width = 256
```

```
    .Height = 256
```

```
    .ControlID = "Exontrol.ChartView"
```

```
    .Create()
```

```
  With .Object
```

```
    .BeginUpdate()
```

```
    .BackColor = RGB(255, 255, 255)
```

```
    .Appearance = 2
```

```
    .HasButtons = 3
```

```
    .ButtonsAlign = 0
```

```
    .PenWidthLink = 3
```

```
  With .Root
```

```
    .Caption = "Root"
```

```
Some information here.
```

```
Line 1:1
```

```
Line 2:2"
```

```
  .Image = 1
```

```
  .AddAssistant("Assistant node")
```

```
End With
```

```
With .Nodes
```

```
  With .Add("Item 1", , "Key1")
```

```
    .HasButton = False
```

```
    .LinkTo = "Key2"
```

```
  End With
```

```
  .Add("SubItem 1", "Key1")
```

```
  With .Add("Sub Item 2", , "Key2")
```

```
    Dim s As String
```

```
    s =
```

```
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCAbigbEsWGAIGA7Eo7HcbLowlpFHZQkZQKA7IspIerIBl
```

```
s = s +
```

```
"fFSEBhikGxSDKbgnglBgoCAAQ7F6IxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE'
```

```
.Picture = s
```

```
.Expanded = False
```

```
.ArrangeSiblingNodesAs = 1
```

```
End With
```

```
.Add("SubItem 1", "Key2")
```

```
.Add("SubItem 2", "Key2")
```

```
End With
```

```
.EndUpdate()
```

```
End With
```

```
End With
```

```
End With
```

```
AxExMenu1.CtlRefresh()
```

The following C# sample adds the Exontrol.ChartView component:

```
EXMENULib.Menu items = axExMenu1.Items;
```

```
EXMENULib.Control control = items.Add(" ActiveX ",
```

```
EXMENULib.ItemTypeEnum.SubControl, 1234).SubControl;
```

```
control.Width = 256;
```

```
control.Height = 256;
```

```
control.ControlID = "Exontrol.ChartView";
```

```
control.Create();
```

```
EXORGCHARTLib.ChartView chart = control.Object as EXORGCHARTLib.ChartView;
```

```
if (chart != null)
```

```
{
```

```
chart.BeginUpdate();
```

```
chart.BackColor = ToUInt32(Color.White);
```

```
chart.Appearance = EXORGCHARTLib.AppearanceEnum.Sunken;
```

```
chart.HasButtons = EXORGCHARTLib.ExpandButtonEnum.exWPlus;
```

```
chart.ButtonsAlign = EXORGCHARTLib.PictureDisplayEnum.UpperLeft;
```

```
chart.PenWidthLink = 3;
```

```
EXORGCHARTLib.Node node = chart.Root;
```

```
node.Caption = "Root
```

Some information here.

```
Line 1:1
```

```
Line 2:2";
```

```
node.Image = 1;
```

```
node.AddAssistant ("Assistant node", null, null);
```

```
EXORGCHARTLib.Nodes nodes = chart.Nodes;
```

```
EXORGCHARTLib.Node node1 = nodes.Add("Item 1", null, "Key1", null, null);
```

```
node1.HasButton = false;
```

```
node1.LinkTo = "Key2";
```

```
nodes.Add("SubItem 1", "Key1", "Key3", null, null);
```

```
EXORGCHARTLib.Node node2 = nodes.Add("Sub Item 2", null, "Key2", null, null);
```

```
String s =
```

```
"gBHJJGHA5MIgAEIe4AAAFhwQiAbCAbigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBl
```

```
s = s +
```

```
"fFSEBhikGxSDKbgnglBgoCAAQ7F6IxoACDRCDwAlwg8SxsAqAYHAQWggAGDgaGAKxEgE"
```

```
node2.Picture = s;
```

```
node2.Expanded = false;
```

```
node2.ArrangeSiblingNodesAs = EXORGCHARTLib.ArrangeSiblingEnum.exHorizontally;
```

```
nodes.Add("SubItem 1", "Key2", "Key4", null, null);
```

```
nodes.Add("SubItem 2", "Key2", "Key5", null, null);
```

```
chart.EndUpdate();
```

```
}
```

```
axExMenu1.CtlRefresh();
```

The C# sample requires a new reference to the Exontrol's ExOrgChart component. Select the Project\Add Reference... and Select COM\ExOrgChart 1.0 Control Library. Once that the component is referred, the EXORGCHARTLib namespace is created, where we can find all objects and definitions for the component being inserted.

The ToUInt32 function converts a Color expression to OLE_COLOR:

```
private UInt32 ToUInt32(Color c)
```

```
{
```

```
long i;
```

```
i = c.R;
```

```

i = i + 256 * c.G;
i = i + 256 * 256 * c.B;
return Convert.ToUInt32(i);
}

```

The following VFP sample adds the Exontrol.ChartView component:

```

With thisform.ExMenu1.Items
  With .Add(" ActiveX ", 3, 1234).SubControl &&
    EXMENUMLibCtl.ItemTypeEnum.SubControl
      .Width = 256
      .Height = 256
      .ControlID = "Exontrol.ChartView"
      .Create
    With .Object
      .BeginUpdate
      .BackColor = RGB(255, 255, 255)
      .Appearance = 2
      .HasButtons = 3
      .ButtonsAlign = 0
      .PenWidthLink = 3
    With .Root
      .Caption = "Root"

```

Some information here.

Line 1:**1**

Line 2:2"

```

      .Image = 1
      .AddAssistant ("Assistant node")

```

EndWith

With .Nodes

```

  With .Add("Item 1", , "Key1")

```

```

    .HasButton = .f

```

```

    .LinkTo = "Key2"

```

EndWith

```

  .Add("SubItem 1", "Key1")

```

```

  With .Add("Sub Item 2", , "Key2")

```

```

    local s

```

s =

"gBHJJGHA5MIgAEIe4AAAFhwQiAbCAbigbEsWGAIGA7Eo7HcbIowlpFHZQkZQKA7IspIErIBl

s = s +

"Fw2HxGJxWLxmNx2PyGRyWTymVy2XzGZzWbzmdz2f0Gh0WjzsMAAhfIEAMMf4AFmmhO

s = s +

"GlPFAkEQhEgLBYhmYgDAWBhCBsFh8HgQJCASCYafYcjQCGBAIBgKhCCwZB6kAAgFgkOBQ

s = s +

"8TAmlYDBigMAgtAyXQyiASJzmqA4CEEf5VIAIRim8XwiiPDRzmsQUKhQLAsEqEBJhCAxSBwI

s = s +

"wAlwg8SxsAqAYHAQWggAGDgaGAKxEgETIzECOoxkqeoAgUFwiHgbQggKHhwBGAYJHIU

s = s +

"mZh8GCBYSDoUBPjMAoansTAGBcVA4AEPINAQOAAEaDREAIYQCgWFATmgO5HI0GAvCeC

s = s +

"GKoSHAHluiGCZGASYymhgagTk8OIQjclBvCqHsNwdRAABAaOAKAwwyAtE4LAboqVnOmI

.Picture = s

.Expanded = .f.

.ArrangeSiblingNodesAs = 1

EndWith

.Add("SubItem 1", "Key2")

.Add("SubItem 2", "Key2")

EndWith

.EndUpdate

EndWith

EndWith

EndWith

thisform.ExMenu1.Object.Refresh

property Item.SubMenu as Menu

Retrieves a Menu object that indicates the item's sub menu. Retrieves Nothing, if the item contains no sub menu.

Type	Description
Menu	A Menu object that contains the submenu items

If the item contains no sub menu, the [Popup](#) property retrieves False. If the Popup retrieves True, the SubMenu property gets the collection of items in the sub menu. Use [Add](#) method to add new items to menu. Use the [VisibleItemsCount](#) property to specify whether the sub menu displays the scroll buttons.

The following VB sample adds a submenu:

```
With ExMenu1
  With .Items
    With .Add("Popup", EXMENULibCtl.ItemTypeEnum.SubMenu).SubMenu
      .Add "Child 1"
    End With
  End With
  .Refresh
End With
```

The following VB sample adds a submenu:

```
With ExMenu1
  With .Items
    With .Add("Popup")
      .Popup = True
      With .SubMenu
        .Add "Child 1"
      End With
    End With
  End With
  .Refresh
End With
```

The following C++ sample adds a submenu:

```
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
CItem item = m_menu.GetItems().Add( "Popup", COleVariant( long(2) /*SubMenu*/ ),
COleVariant( (long)1234 ) );
item.GetSubMenu().Add( "Child 1", vtMissing, vtMissing );
m_menu.Refresh();
```

The following C++ sample adds a submenu:

```
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;
CItem item = m_menu.GetItems().Add( "Popup", vtMissing, COleVariant( (long)1234 ) );
item.SetPopup( TRUE );
item.GetSubMenu().Add( "Child 1", vtMissing, vtMissing );
m_menu.Refresh();
```

The following VB.NET sample adds a submenu:

```
With AxExMenu1.Items
    With .Add("Popup", EXMENU.Lib.ItemTypeEnum.SubMenu, 1234).SubMenu
        .Add("Child")
    End With
End With
AxExMenu1.CtlRefresh()
```

The following VB.NET sample adds a submenu:

```
With AxExMenu1.Items
    Dim i As EXMENU.Lib.item = .Add("Popup", , 1234)
    i.Popup = True
    With i.SubMenu
        .Add("Child")
    End With
End With
AxExMenu1.CtlRefresh()
```

The following C# sample adds a submenu:

```
EXMENU.Lib.item item = axExMenu1.Items.Add("Popup",
EXMENU.Lib.ItemTypeEnum.SubMenu, 1234);
EXMENU.Lib.Menu subMenu = item.SubMenu;
```

```
subMenu.Add("Child 1", null, null);  
axExMenu1.CtlRefresh();
```

The following C# sample adds a submenu:

```
EXMENUMLib.item item = axExMenu1.Items.Add("Popup", null, 1234);  
item.Popup = true;  
EXMENUMLib.Menu subMenu = item.SubMenu;  
subMenu.Add("Child 1", null, null);  
axExMenu1.CtlRefresh();
```

The following VFP sample adds a submenu:

```
With thisform.ExMenu1.Items  
  with .Add("Popup", 2, 1234).SubMenu  
    .Add("Child")  
  endwith  
EndWith  
thisform.ExMenu1.Object.Refresh
```

The following VFP sample adds a submenu:

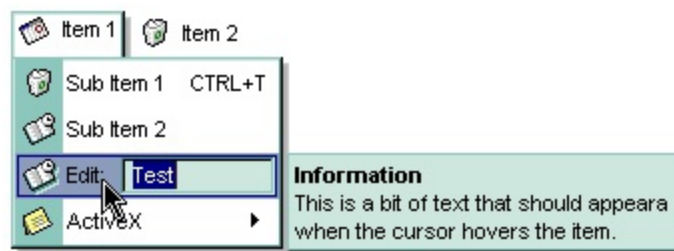
```
With thisform.ExMenu1.Items  
  with .Add("Popup", , 1234)  
    .Popup = .t.  
    With .SubMenu  
      .Add("Child")  
    EndWith  
  endwith  
EndWith  
thisform.ExMenu1.Object.Refresh
```

property Item.ToolTip as Variant

Specifies the item's tooltip.

Type	Description
Variant	A string expression that indicates the item's tooltip.

The control displays the item's tooltip when the mouse is over the item. Use the [ToolTipTitle](#) property to define the title for the item's tooltip. Use the [ToolTipDelay](#) property to specify the time in ms that passes before the ToolTip appears. By default, the ToolTip property is empty. If the ToolTip property is empty the control displays no tooltip when cursor is over the item. Use the [ToolTipWidth](#) property to specify the width of the item's 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 [ToolTipFont](#) property to assign a different font for tooltips.



The ToolTip supports the following built-in HTML tags:

- ** bold **
- **<u> underline </u>**
- **<s> strikeout </s>**
- **<i> italic </i>**
- **<fgcolor = FF0000> fgcolor </fgcolor>**
- **<bgcolor = FF0000> bgcolor </bgcolor>**
- **
** breaks a line.
- **<solidline>** draws a solid line
- **<dotline>** draws a dotted line
- **<upline>** draws the line to the top of the text line
- **<r>** aligns the rest of the text line to the right side.
- **number[:width]** inserts an icon inside the cell's caption. The number indicates the index of the icon being inserted. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels.
- **key[:width]** inserts a custom size picture being loaded using the

[HTMLPicture](#) property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used.

- **text ** displays portions of text with a different font and/or different size. For instance, the `bit` draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, `bit` displays the bit text using the current font, but with a different size.

Newer HTML format supports subscript and superscript like follows:

- **<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}

Also, newer HTML format supports decorative text like follows:

- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or `<fgcolor>` defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The `` HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The `<gra>` with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the `"<gra FFFFFFF;1;1>gradient-center</gra>"` generates the following picture:

gradient-center

- **<out rrggbb;width> ... </out>** shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray, width indicates the size of the outline, 1 if missing. The text color or `<fgcolor>` defines the color to show the inside text. The `` HTML tag can be used to define the height of the font. For instance the `"<out 000000><fgcolor=FFFFFF>outlined</fgcolor></out>"` generates the following picture:

outlined

- `<sha rrggbb;width;offset> ... </sha>` define a text with a shadow, where rr/gg/bb represents the red/green/blue values of the shadow color, 808080 if missing as gray, width indicates the size of shadow, 4 if missing, and offset indicates the offset from the origin to display the text's shadow, 2 if missing. The text color or `<fgcolor>` defines the color to show the inside text. The `` HTML tag can be used to define the height of the font. For instance the "`<sha>shadow</sha>`" generates the following picture:

shadow

or "`<sha 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></sha>`" gets:

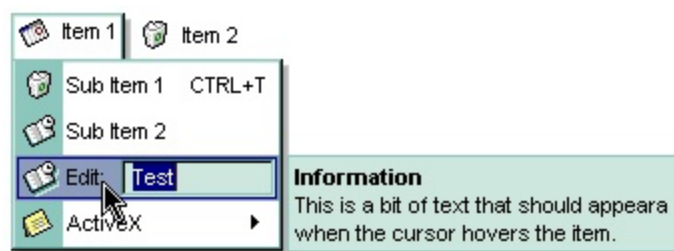
outline anti-aliasing

property Item.ToolTipTitle as Variant

Specifies the title of the item's tooltip.

Type	Description
Variant	A string expression that defines the title for the item's tooltip.

Use the ToolTipTitle property to define the title for the item's tooltip. Use the [ToolTip](#) property to assign a tooltip to an item. Use the [ToolTipDelay](#) property to specify the time in ms that passes before the ToolTip appears. By default, the ToolTip property is empty. If the ToolTip property is empty the control displays no tooltip when cursor is over the item. Use the [ToolTipWidth](#) property to specify the width of the item's tooltip window.



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 appears as underlined.

Use the [Bold](#), [Italic](#), [Underline](#) and [Strikeout](#) properties to customize the font attribute for any menu item. Use the <u> HTML tag in the [Caption](#) property to specify that parts of the caption appear as underlined. Use the [Font](#) property to specify the control's font.

The following VB sample underlines the item with the identifier 50:

```
With ExMenu1.Item(50)
    .Underline = True
End With
```

The following C++ sample underlines the item with the identifier 50:

```
CItem item = m_menu.GetItem( COleVariant( long( 50 ) ) );
item.SetUnderline( TRUE );
```

The following VB.NET sample underlines the item with the identifier 50:

```
With AxExMenu1.item(50)
    .Underline = True
End With
```

The following C# sample underlines the item with the identifier 50:

```
EXMENUlib.item item = axExMenu1[50];
item.Underline = true;
```

The following VFP sample underlines the item with the identifier 50:

```
With thisform.ExMenu1.Item(50)
    .Underline = .t.
EndWith
```

property Item.UseBackColor as Boolean

Retrieves or sets a value that indicates whether the item's background color is specified by BackColor property.

Type	Description
Boolean	A boolean expression that indicates whether the item's background color is specified by BackColor property.

The UseBackColor property specifies whether the [BackColor](#) property has effect. Changing the BackColor property automatically sets the UseBackColor property on True.



The following VB sample changes the background color for the item with the identifier 50:

```
With ExMenu1.Item(50)
    .BackColor = vbRed
End With
```

The following C++ sample changes the background color for the item with the identifier 50:

```
CItem item = m_menu.GetItem( COleVariant( long( 50 ) ) );
item.SetBackColor( RGB(255,0,0) );
```

The following VB.NET sample changes the background color for the item with the identifier 50:

```
With AxExMenu1.item(50)
    .BackColor = ToUInt32(Color.Red)
End With
```

where the ToUInt32 function converts a Color expression to OLE_COLOR:

```
Shared Function ToUInt32(ByVal c As Color) As UInt32
    Dim i As Long
    i = c.R
```

```
i = i + 256 * c.G  
i = i + 256 * 256 * c.B  
ToUInt32 = Convert.ToUInt32(i)  
End Function
```

The following C# sample changes the background color for the item with the identifier 50:

```
EXMENUlib.item item = axExMenu1[50];  
item.BackColor = ToUInt32(Color.Red);
```

where the ToUInt32 function converts a Color expression to OLE_COLOR:

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

The following VFP sample changes the background color for the item with the identifier 50:

```
With thisform.ExMenu1.Item(50)  
    .BackColor = RGB(255,0,0)  
EndWith
```

property Item.UseForeColor as Boolean

Retrieves or sets a value that indicates whether the item's foreground color is specified by ForeColor property.

Type	Description
Boolean	A boolean expression that indicates whether the item's foreground color is specified by ForeColor property.

The UseBackColor property specifies whether the [ForeColor](#) property has effect. The UseForeColor property is set on True, when the user changes the ForeColor property.



The following VB sample changes the foreground color for the item with the identifier 50:

```
With ExMenu1.Item(50)
    .ForeColor = vbRed
End With
```

The following C++ sample changes the foreground color for the item with the identifier 50:

```
CItem item = m_menu.GetItem( COleVariant( long( 50 ) ) );
item.SetForeColor( RGB(255,0,0) );
```

The following VB.NET sample changes the foreground color for the item with the identifier 50:

```
With AxExMenu1.item(50)
    .ForeColor = ToUInt32(Color.Red)
End With
```

where the ToUInt32 function converts a Color expression to OLE_COLOR:

```
Shared Function ToUInt32(ByVal c As Color) As UInt32
    Dim i As Long
    i = c.R
```

```
i = i + 256 * c.G
i = i + 256 * 256 * c.B
ToUInt32 = Convert.ToUInt32(i)
End Function
```

The following C# sample changes the foreground color for the item with the identifier 50:

```
EXMENUMLib.item item = axExMenu1[50];
item.ForeColor = ToUInt32(Color.Red);
```

where the ToUInt32 function converts a Color expression to OLE_COLOR:

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

The following VFP sample changes the foreground color for the item with the identifier 50:

```
With thisform.ExMenu1.Item(50)
    .ForeColor = RGB(255,0,0)
EndWith
```

property Item.UserData as Variant

Associates an extra data to the object.

Type	Description
Variant	A Variant that specifies the item's user data.

The UserData property associates an extra data to the item. The UserData property is not used by the control. The UserData are of Variant type, so you will be able to save here what ever you want: numbers, objects, strings, and so on

property Item.Visible as Boolean

Specifies whether the item is visible or hidden.

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

By default, all items are visible. Use the Visible property to hide an item. Use the [Enabled](#) property to disable an item. Use the [ForeColor](#) property to specify the item's foreground color. Use the [BackColor](#) property to specify the item's background color. Use the [Separator](#) property to specify whether an item is a separator item. Use the [Refresh](#) method to refresh the control.

The following VB sample hides the item with the identifier 10:

```
With ExMenu1.Item(10)
    .Visible = False
End With
ExMenu1.Refresh
```

The following C++ sample hides the item with the identifier 10:

```
CItem item = m_menu.GetItem( COleVariant( long( 10 ) ) );
item.SetVisible( FALSE );
m_menu.Refresh();
```

The following VB.NET sample hides the item with the identifier 10:

```
With AxExMenu1.item(10)
    .Visible = False
End With
AxExMenu1.CtlRefresh()
```

The following C# sample hides the item with the identifier 10:

```
EXMENULib.item item = axExMenu1[10];
item.Visible = false;
axExMenu1.CtlRefresh();
```

The following VFP sample hides the item with the identifier 10:

```
With thisform.ExMenu1.Item(10)
```

```
  .Visible = .f
```

```
EndWith
```

```
thisform.ExMenu1.Object.Refresh
```

Menu object

The Menu object holds a collection of [Item](#) objects. Use the [Items](#) property to access to menu bar items collection. The Menu object supports the following properties and methods:

Name	Description
Add	Adds a new item to menu and retrieves the newly created object.
Clear	Clear the menu items.
Count	Counts the items.
Cursor	Specifies the shape of the cursor when mouse hovers the menu.
Find	Searches recursively for an item given its identifier or its caption. The method searches any submenu items.
item	Returns a specific Item object given its caption or its identifier.
ItemByIndex	Returns a specific Item object given its caption or its index.
Picture	Retrieves or sets the background's picture.
PictureDisplay	Retrieves or sets a value that indicates the way how the picture is displayed.
Remove	Removes the item given its caption or its identifier.
ToString	Saves or loads the menu from a formatted string.
VisibleItemsCount	Specifies the maximum number of visible items at one time.
Width	Retrieves or sets a value that indicates the width of the menu.

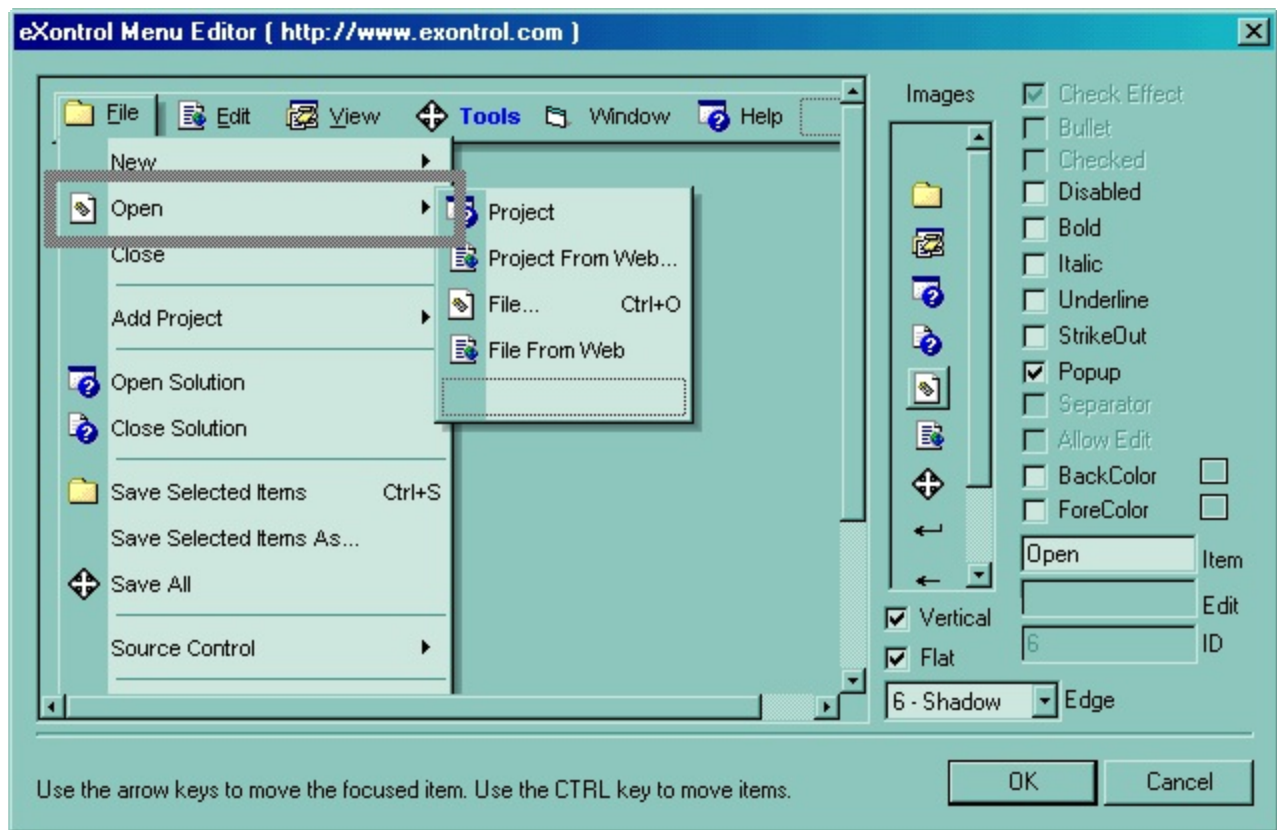
method Menu.Add (Caption as String, [ItemType as Variant], [ID as Variant])

Adds a new item to menu and retrieves the newly created object.

Type	Description
Caption as String	A string expression that defines the item's caption. The caption argument may include built in HTML tags like described in the Caption property.
ItemType as Variant	An ItemTypeEnum expression that indicates the type of the item like follows: 0 - Default, 1 - Separator, 2 - Popup, 3 - SubControl
ID as Variant	A long expression that indicates the item's identifier. If the value is missing, the control automatically assign a new identifier.
Return	Description
Item	An Item object that indicates the newly created menu item.

Use the Add method to add items are runtime. Use the [ToString](#) method to quick load items from a formatted string. The [Separator](#) property specifies whether an item is of Separator type . The [Popup](#) property specifies whether the item contains sub items. Use [SubMenu](#) property to access the sub items of a popup item. The [Refresh](#) method must be called to reflect the changes on the control's content. Use the [SubControl](#) property to specify the control's identifier, when ItemType is SubControl. The [ItemHeight](#) property specifies the height for the items in the menu

The control provides a WYSWYG editor that helps building a menu at design time. Select the control in design mode, and select 'Properties' control's context menu in order to access the menu's editor at design time (like in the following screen shot):



The following VB sample adds few items at runtime:

```
With ExMenu1
```

```
  .Debug = True
```

```
  With .Items
```

```
    With .Add("Menu 1", EXMENU LibCtl.SubMenu).SubMenu
```

```
      .Add("File", EXMENU LibCtl.Default, 1222).Alignment = exRight
```

```
      .Add("Open", EXMENU LibCtl.Default, 1223).Alignment = exRight
```

```
      .Add "Print Preview", EXMENU LibCtl.Default, 1224
```

```
      .Add "", EXMENU LibCtl.Separator
```

```
      .Add("Close", , 1225).Image = 1
```

```
    End With
```

```
  End With
```

```
  .Refresh
```

```
End With
```

The following VB sample adds few items using built-in HTML tags:

```
With ExMenu1
```

```
  With .Items
```

```
    With .Add("Menu <fgcolor=0000FF> 1 </fgcolor>", EXMENU LibCtl.SubMenu, 1221)
```

```

With .SubMenu
    .Add "File  <b>CTRL+F</b>", EXMENUlibCtl.Default, 1222
    .Add("Open", EXMENUlibCtl.Default, 1223).Alignment = exRight
    .Add "<fgcolor=00FFFF>Print</fgcolor> <u>Preview</u>",
EXMENUlibCtl.Default, 1224
    .Add "", EXMENUlibCtl.Separator
    .Add("Close", , 1225).Image = 1
End With
End With
End With
.AddAcelerator 1222, KeyCodeConstants.vbKeyF, True, False, False
.Refresh
End With

```

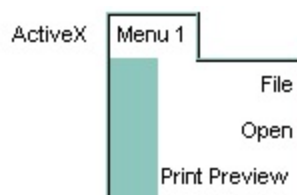
Using built-in HTML tags you can colorize your items.

The following VB changes the item with the identifier 7 to host an ActiveX control (A Microsoft Calendar Control in this case):

```

With ExMenu1(7)
    .Control = True
    With .SubControl
        .ControlID = "MSCAL.Calendar"
        .Width = 176
        .Height = 156
        .Create
        With .Object
            .ShowDateSelectors = False
        End With
    End With
End With
End With

```



The following VB sample adds some items that are aligned to the right:

```
With ExMenu1
```

```
.Border = BumpBorder
```

```
With .Items
```

```
With .Add("Menu 1", EXMENUMLibCtl.SubMenu).SubMenu
```

```
.Add("File").Alignment = exRight
```

```
.Add("Open").Alignment = exRight
```

```
.Add ("Print Preview")
```

```
End With
```

```
End With
```

```
.Refresh
```

```
End With
```

The following C++ sample adds some items that are aligned to the right:

```
COleVariant vtMissing; V_VT( &vtMissing ) = VT_ERROR;  
m_menu.SetBorder( 5/*BumpBorder*/ );  
CItem item = m_menu.GetItems().Add( "Menu 1", COleVariant( (long)2 /*SubMenu*/ ),  
vtMissing );  
CItem itemF = item.GetSubMenu().Add( "File", vtMissing, vtMissing );  
itemF.SetAlignment( 2 /*RightAlignment*/ );  
CItem itemO = item.GetSubMenu().Add( "Open", vtMissing, vtMissing );  
itemO.SetAlignment( 2 /*RightAlignment*/ );  
item.GetSubMenu().Add("Print Preview", vtMissing, vtMissing);  
m_menu.Refresh();
```

The following VB.NET sample adds some items that are aligned to the right:

```
With AxExMenu1
```

```
.Border = EXMENUMLib.BorderEnum.BumpBorder
```

```
With .Items
```

```
With .Add("Menu 1", EXMENUMLib.ItemTypeEnum.SubMenu).SubMenu
```

```
.Add("File").Alignment = EXMENUMLib.AlignmentEnum.exRight
```

```
.Add("Open").Alignment = EXMENUMLib.AlignmentEnum.exRight
```

```
.Add("Print Preview")
```

```
End With
```

```
End With
```

```
.CtlRefresh()
```

```
End With
```

The following C# sample adds some items that are aligned to the right:

```
axExMenu1.Border = EXMENUMLib.BorderEnum.BumpBorder;
EXMENUMLib.Menu items = axExMenu1.Items;
EXMENUMLib.Menu subMenu = items.Add("Menu 1", EXMENUMLib.ItemTypeEnum.SubMenu,
null).SubMenu;
subMenu.Add("File", null, null).Alignment = EXMENUMLib.AlignmentEnum.exRight;
subMenu.Add("Open", null, null).Alignment = EXMENUMLib.AlignmentEnum.exRight;
subMenu.Add("Print Preview", null, null);
axExMenu1.CtlRefresh();
```

The following VFP sample adds some items that are aligned to the right:

```
With thisform.ExMenu1
  .Border = 5 && BumpBorder
  With .Items
    With .Add("Menu 1", 2).SubMenu && EXMENUMLibCtl.SubMenu
      .Add("File").Alignment = 2 && exRight
      .Add("Open").Alignment = 2 && exRight
      .Add ("Print Preview")
    EndWith
  EndWith
EndWith
thisform.ExMenu1.Object.Refresh
```

method Menu.Clear ()

Clear the menu items.

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

The Clear method clears the items in the menu . Use the [Remove](#) method to remove an item. Use the [Refresh](#) method if you are clearing the menu bar items. Use the [SubMenu](#) property to access the sub menu. Use the [Visible](#) property to hide an item. Use the [PopupBackColor](#) property to specify the background color for the drop down menus. Use the [ItemHeight](#) property to specify the height for the items. Use the [Items](#) property to retrieve the items in the menu bar. Use the [Images](#) method to load icons at runtime.

The following VB sample clears all items in the control:

```
With ExMenu1.Items  
    .Clear  
End With
```

The following C++ sample clears all items in the control:

```
m_menu.GetItems().Clear();
```

The following VB.NET sample clears all items in the control:

```
With AxExMenu1.Items  
    .Clear()  
End With
```

The following C# sample clears all items in the control:

```
axExMenu1.Items.Clear();
```

The following VFP sample clears all items in the control:

```
With thisform.ExMenu1.Items  
    .Clear()  
EndWith
```

property Menu.Count as Long

Counts the items.

Type	Description
Long	A long expression that indicates the count of items

The Count property gets the number of items in the collection. Use the [ItemByIndex](#) property to retrieve an item by its index. Use the [Item](#) property to access an item giving its identifier. Use the [ID](#) property to specify the item's identifier. Use the [Find](#) property to look for an item. Use the [Caption](#) property to specify the caption of the item. Use the [VisibleItemsCount](#) property to specify the number of visible items.

The following VB sample enumerates the items in a sub menu:

```
With ExMenu1(10).SubMenu
    Dim i As Long
    For i = 0 To .Count - 1
        Debug.Print .ItemByIndex(i).Caption
    Next
End With
```

The following C++ sample enumerates the items in a sub menu:

```
CMenu1 subMenu = m_menu.GetItem( COleVariant( long(10) ) ).GetSubMenu();
for ( long i = 0; i < subMenu.GetCount(); i++ )
{
    CItem item = subMenu.GetItemByIndex( COleVariant( i ) );
    OutputDebugString( item.GetCaption() );
}
```

The following VB.NET sample enumerates the items in a sub menu:

```
With AxExMenu1(10).SubMenu
    Dim i As Long
    For i = 0 To .Count - 1
        Debug.WriteLine(.ItemByIndex(i).Caption())
    Next
End With
```

The following C# sample enumerates the items in a sub menu:

```
EXMENUMLib.Menu subMenu = axExMenu1[10].SubMenu;
for (int i = 0; i < subMenu.Count; i++)
{
    EXMENUMLib.item item = subMenu.get_ItemByIndex(i);
    System.Diagnostics.Debug.WriteLine(item.Caption);
}
```

The following VFP sample enumerates the items in a sub menu:

```
With thisform.ExMenu1.Item(10).SubMenu
    local i
    For i = 0 To .Count - 1
        wait window nowait .ItemByIndex(i).Caption
    Next
EndWith
```

property Menu.Cursor as Variant

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

Type	Description
Variant	A string expression that indicates a predefined value listed below, a string expression that indicates the path to a cursor file, a long expression that indicates the handle of the cursor.

Use the [Cursor](#) property to specify the shape of the cursor when the mouse pointer hovers a sub menu. Use the [Cursor](#) property to specify the cursor that control displays when the mouse pointer hovers the item. Use the [Cursor](#) property to specify the shape of the cursor when the mouse pointer hovers an item.

Here's the list of predefined values (string expressions):

- **"exDefault"** - (Default) Shape determined by the object.
- **"exArrow"** - Arrow.
- **"exCross"** - Cross (cross-hair pointer).
- **"exIBeam"** - I Beam.
- **"exIcon"** - Icon (small square within a square).
- **"exSize"** - Size (four-pointed arrow pointing north, south, east, and west).
- **"exSizeNESW"** - Size NE SW (double arrow pointing northeast and southwest).
- **"exSizeNS"** - Size N S (double arrow pointing north and south).
- **"exSizeNWSE"** - Size NW, SE.
- **"exSizeWE"** - Size W E (double arrow pointing west and east).
- **"exUpArrow"** - Up Arrow.
- **"exHourglass"** - Hourglass (wait).
- **"exNoDrop"** - No Drop.
- **"exArrowHourglass"** - Arrow and hourglass.
- **"exHelp"** - Arrow and question mark.
- **"exSizeAll"** - Size all.

If the cursor value is a string expression, the control looks first if it is not a predefined value like listed above, and if not, it tries to load the cursor from a file. If the Cursor property is a long expression it always indicates a handle to a cursor. The API functions like: LoadCursor or LoadCursorFromFile retrieves a handle to a cursor. In .NET framework, the Handle parameter of the Cursor object specifies the handle to the cursor. Use the Cursors object to access to the list of predefined cursors in the .NET framework.

property Menu.Find (ID as Variant) as Item

Searches recursively for an item given its identifier or its caption. The method searches any submenu items.

Type	Description
ID as Variant	A long expression that indicates the item's identifier, or a string expression that indicates the item's caption searched for.
Item	An Item object that defines the menu item.

The Find method searches the submenus as well.

property Menu.item (ID as Variant) as Item

Returns a specific Item object given its caption or its identifier.

Type	Description
ID as Variant	A long expression that indicates the item's identifier, or a string expression that indicates the item's caption searched for.
Item	An Item object that defines the menu item.

Use the Item property to retrieve the item giving its identifier. The Item property doesn't look recursively for the item. Use the [Find](#) property to look recursively for an item giving its caption or its identifier. Use the [ItemByIndex](#) property to retrieve an item giving its index. Use the [Item](#) property to retrieve an item from the menu (it looks recursively if case).

The following VB sample recursively enumerates all items in the menu:

```
Private Sub scan(ByVal e As EXMENULibCtl.Menu)
  If Not (e Is Nothing) Then
    Dim i As EXMENULibCtl.Item
    For Each i In e
      Debug.Print i.Caption
      scan i.SubMenu
    Next
  End If
End Sub
```

Or an alternative function without using for each statement will be:

```
Private Sub scan(ByVal e As EXMENULibCtl.Menu)
  If Not (e Is Nothing) Then
    Dim j As Long
    For j = 0 To e.Count - 1
      Debug.Print e.ItemByIndex(j).Caption
      scan e.ItemByIndex(j).SubMenu
    Next
  End If
End Sub
```


property Menu.ItemByIndex (Index as Variant) as Item

Returns a specific Item object given its caption or its index.

Type	Description
Index as Variant	A long expression that indicates the index of item in the collection, a string expression that indicates the item's caption.
Item	An Item object being searched.

Use the ItemByIndex property to access an Item object by its caption. Use the [Count](#) property to specify the number of items in the sub menu. Use the [Item](#) property to access an item by its identifier.

The following VB sample enumerates the items in a sub menu:

```
With ExMenu1(10).SubMenu
  Dim i As Long
  For i = 0 To .Count - 1
    Debug.Print .ItemByIndex(i).Caption
  Next
End With
```

The following VB sample recursively enumerates all items in the menu:

```
Private Sub scan(ByVal e As EXMENULibCtl.Menu)
  If Not (e Is Nothing) Then
    Dim i As EXMENULibCtl.Item
    For Each i In e
      Debug.Print i.Caption
      scan i.SubMenu
    Next
  End If
End Sub
```

Or an alternative function without using for each statement will be:

```
Private Sub scan(ByVal e As EXMENULibCtl.Menu)
  If Not (e Is Nothing) Then
    Dim j As Long
```

```

For j = 0 To e.Count - 1
    Debug.Print e.ItemByIndex(j).Caption
    scan e.ItemByIndex(j).SubMenu
Next
End If
End Sub

```

The following C++ sample enumerates the items in a sub menu:

```

CMenu1 subMenu = m_menu.GetItem( COleVariant( long(10) ) ).GetSubMenu();
for ( long i = 0; i < subMenu.GetCount(); i++ )
{
    CItem item = subMenu.GetItemByIndex( COleVariant( i ) );
    OutputDebugString( item.GetCaption() );
}

```

The following VB.NET sample enumerates the items in a sub menu:

```

With AxExMenu1(10).SubMenu
    Dim i As Long
    For i = 0 To .Count - 1
        Debug.WriteLine(.ItemByIndex(i).Caption())
    Next
End With

```

The following C# sample enumerates the items in a sub menu:

```

EXMENUMLib.Menu subMenu = axExMenu1[10].SubMenu;
for (int i = 0; i < subMenu.Count; i++)
{
    EXMENUMLib.item item = subMenu.get_ItemByIndex(i);
    System.Diagnostics.Debug.WriteLine(item.Caption);
}

```

The following VFP sample enumerates the items in a sub menu:

```

With thisform.ExMenu1.Item(10).SubMenu
    local i
    For i = 0 To .Count - 1

```

wait window nowait .ItemByIndex(i).Caption

Next

EndWith

property Menu.Picture as IPictureDisp

Retrieves or sets the background's picture.

Type	Description
IPictureDisp	A Picture object being displayed on the menu's background.

Use the [Picture](#) property to put a picture on the menu's background. Use the [PictureDisplay](#) property to align picture on the menu's background. Use the [Picture](#) property to display a picture on the control's background. Use the [BackColor](#) property to specify the item's background color. Use the [ForeColor](#) property to specify the item's foreground color. Use the [Caption](#) property to specify the caption of the item. Use the [Picture](#) property to specify a picture on the control's background (menu bar). Use the [PopupBackColor](#) property to change the background color for the popup menu.



The following VB sample assigns a picture to a submenu:

```
With ExMenu1(10).SubMenu
    .PictureDisplay = Tile
    .Picture = LoadPicture("c:\winnt\zapotec.bmp")
End With
```

The following C++ sample assigns a picture to a submenu:

```
IPictureDisp* pPicture = NULL;
if ( LoadPicture( "c:\\winnt\\zapotec.bmp", &pPicture ) )
{
    CMenu1 subMenu = m_menu.GetItem( COleVariant(long(10)) ).GetSubMenu();
    subMenu.SetPicture( pPicture );
    subMenu.SetPictureDisplay( 48 /*Tile*/ );
}
```

where the LoadPicture function gets the IPictureDisp from a file:

```

#include
BOOL LoadPicture( LPCTSTR szFileName, IPictureDisp** ppPictureDisp )
{
    BOOL bResult = FALSE;
    if ( szFileName )
    {
        OFSTRUCT of;
        HANDLE hFile = NULL;;
#ifdef _UNICODE
        USES_CONVERSION;
        if ( (hFile = (HANDLE)OpenFile( W2A(szFileName), &of,, OF_READ |
OF_SHARE_COMPAT)) != (HANDLE)HFILE_ERROR )
#else
        if ( (hFile = (HANDLE)OpenFile( szFileName, &of,, OF_READ | OF_SHARE_COMPAT)) !=
(HANDLE)HFILE_ERROR )
#endif
        {
            *ppPictureDisp = NULL;
            DWORD dwHighWord = NULL, dwSizeLow = GetFileSize( hFile, &dwHighWord );
            DWORD dwFileSize = dwSizeLow;
            HRESULT hResult = NULL;
            if ( HGLOBAL hGlobal = GlobalAlloc(GMEM_MOVEABLE, dwFileSize) )
                if ( void* pvData = GlobalLock( hGlobal ) )
                {
                    DWORD dwReadBytes = NULL;
                    BOOL bRead = ReadFile( hFile, pvData, dwFileSize, &dwReadBytes,, NULL );
                    GlobalUnlock( hGlobal );
                    if ( bRead )
                    {
                        CComPtr spStream;
                        _ASSERT( dwFileSize == dwReadBytes );
                        if ( SUCCEEDED( CreateStreamOnHGlobal( hGlobal, TRUE, &spStream; ) ) )
                            if ( SUCCEEDED( hResult = OleLoadPicture( spStream, 0, FALSE,
IID_IPictureDisp, (void**)ppPictureDisp ) ) )
                                bResult = TRUE;
                    }
                }
        }
    }
}

```

```

        CloseHandle( hFile );
    }
}
return bResult;
}

```

The following VB.NET sample assigns a picture to a submenu:

```

With AxExMenu1.item(10).SubMenu
    .Picture = IPDH.GetIPictureDisp(Image.FromFile("c:\winnt\zapotec.bmp"))
    .PictureDisplay = EXMENUMLib.PictureDisplayEnum.Tile
End With

```

where the IPDH class is defined like follows:

```

Public Class IPDH
    Inherits System.Windows.Forms.AxHost

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

    Public Shared Function GetIPictureDisp(ByVal image As Image) As Object
        GetIPictureDisp = AxHost.GetIPictureDispFromPicture(image)
    End Function

End Class

```

The following C# sample assigns a picture to a submenu:

```

EXMENUMLib.Menu subMenu = axExMenu1[10].SubMenu;
subMenu.Picture = IPDH.GetIPictureDisp(Image.FromFile("c:\\winnt\\zapotec.bmp")) as
stdole.IPictureDisp ;
subMenu.PictureDisplay = EXMENUMLib.PictureDisplayEnum.Tile;

```

The following VFP sample assigns a picture to a submenu:

```

With thisform.ExMenu1.Item(10).SubMenu
    .PictureDisplay = 48 && Tile

```

```
.Picture = LoadPicture("c:\WinNT\Zapotec.bmp")
```

```
EndWith
```

property Menu.PictureDisplay as PictureDisplayEnum

Retrieves or sets a value that indicates the way how the picture is displayed.

Type	Description
PictureDisplayEnum	A PictureDisplayEnum expression that specifies how the picture is displayed on the menu's background.

Use the PictureDisplay property to arrange the menu's picture. The PictureDisplay property has no effect if the menu's background has no picture loaded. Use the [Picture](#) property to put a picture on the menu's background. Use the [Picture](#) property to put a picture on the control's background. Use the [BackColor](#) property to specify the item's background color. Use the [ForeColor](#) property to specify the item's foreground color. Use the [Caption](#) property to specify the caption of the item.



The following VB sample assigns a picture to a submenu:

```
With ExMenu1(10).SubMenu
    .PictureDisplay = Tile
    .Picture = LoadPicture("c:\winnt\zapotec.bmp")
End With
```

The following C++ sample assigns a picture to a submenu:

```
IPictureDisp* pPicture = NULL;
if ( LoadPicture( "c:\\winnt\\zapotec.bmp", &pPicture ) )
{
    CMenu1 subMenu = m_menu.GetItem( COleVariant(long(10)) ).GetSubMenu();
    subMenu.SetPicture( pPicture );
    subMenu.SetPictureDisplay( 48 /*Tile*/ );
}
```

where the LoadPicture function gets the IPictureDisp from a file:

```

#include
BOOL LoadPicture( LPCTSTR szFileName, IPictureDisp** ppPictureDisp )
{
    BOOL bResult = FALSE;
    if ( szFileName )
    {
        OFSTRUCT of;
        HANDLE hFile = NULL;;
#ifdef _UNICODE
        USES_CONVERSION;
        if ( (hFile = (HANDLE)OpenFile( W2A(szFileName), &of,, OF_READ |
OF_SHARE_COMPAT)) != (HANDLE)HFILE_ERROR )
#else
        if ( (hFile = (HANDLE)OpenFile( szFileName, &of,, OF_READ | OF_SHARE_COMPAT)) !=
(HANDLE)HFILE_ERROR )
#endif
        {
            *ppPictureDisp = NULL;
            DWORD dwHighWord = NULL, dwSizeLow = GetFileSize( hFile, &dwHighWord; );
            DWORD dwFileSize = dwSizeLow;
            HRESULT hResult = NULL;
            if ( HGLOBAL hGlobal = GlobalAlloc(GMEM_MOVEABLE, dwFileSize) )
                if ( void* pvData = GlobalLock( hGlobal ) )
                {
                    DWORD dwReadBytes = NULL;
                    BOOL bRead = ReadFile( hFile, pvData, dwFileSize, &dwReadBytes,, NULL );
                    GlobalUnlock( hGlobal );
                    if ( bRead )
                    {
                        CComPtr spStream;
                        _ASSERT( dwFileSize == dwReadBytes );
                        if ( SUCCEEDED( CreateStreamOnHGlobal( hGlobal, TRUE, &spStream; ) ) )
                            if ( SUCCEEDED( hResult = OleLoadPicture( spStream, 0, FALSE,
IID_IPictureDisp, (void**)ppPictureDisp ) ) )
                                bResult = TRUE;
                    }
                }
        }
    }
}

```

```

        CloseHandle( hFile );
    }
}
return bResult;
}

```

The following VB.NET sample assigns a picture to a submenu:

```

With AxExMenu1.item(10).SubMenu
    .Picture = IPDH.GetIPictureDisp(Image.FromFile("c:\winnt\zapotec.bmp"))
    .PictureDisplay = EXMENUMLib.PictureDisplayEnum.Tile
End With

```

where the IPDH class is defined like follows:

```

Public Class IPDH
    Inherits System.Windows.Forms.AxHost

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

    Public Shared Function GetIPictureDisp(ByVal image As Image) As Object
        GetIPictureDisp = AxHost.GetIPictureDispFromPicture(image)
    End Function

End Class

```

The following C# sample assigns a picture to a submenu:

```

EXMENUMLib.Menu subMenu = axExMenu1[10].SubMenu;
subMenu.Picture = IPDH.GetIPictureDisp(Image.FromFile("c:\\winnt\\zapotec.bmp")) as
stdole.IPictureDisp ;
subMenu.PictureDisplay = EXMENUMLib.PictureDisplayEnum.Tile;

```

The following VFP sample assigns a picture to a submenu:

```

With thisform.ExMenu1.Item(10).SubMenu
    .PictureDisplay = 48 && Tile

```

```
.Picture = LoadPicture("c:\WinNT\Zapotec.bmp")
```

```
EndWith
```

method **Menu.Remove (ID as Variant)**

Removes the item given its caption or its identifier.

Type	Description
ID as Variant	A long expression that indicates the item's identifier, or a string expression that indicates the item's caption removed

Use [Clear](#) method to clear the items collection.

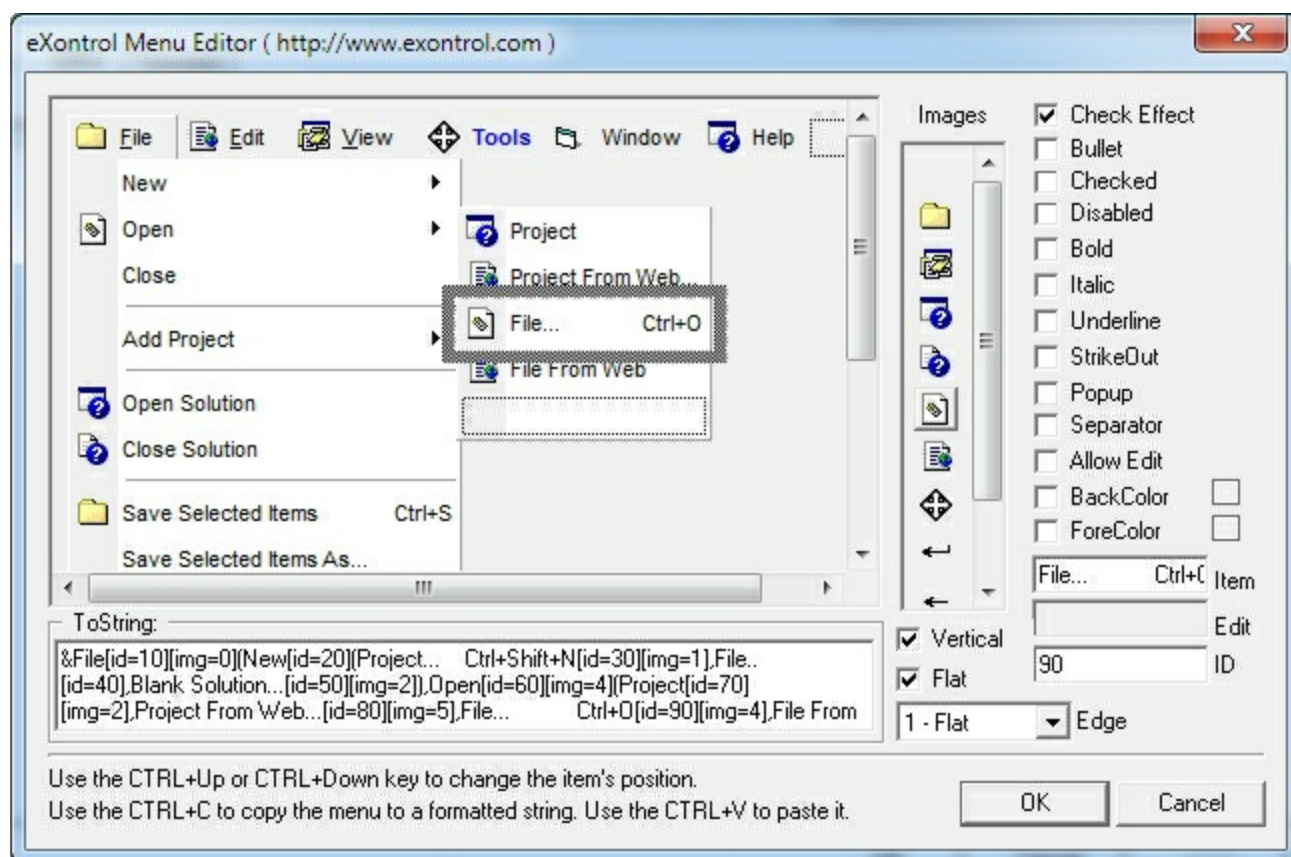
property Menu.ToString as String

Saves or loads the menu from a formatted string.

Type	Description
String	A String expression that indicates the list of items.

Use the ToString method to quick load the items from a formatted string. Use the [Template](#) property to load the control's data using the x-script template. The [ExecuteTemplate](#) property gets the result after executing a template script.

The following screen shot shows how the control's WYSWYG editor looks like:



The ToString's syntax in BNF notation:

```
<TOSTRING> ::= <ITEMS>
<ITEMS> ::= <ITEM>["("<ITEMS>")"][","<ITEMS>]
<ITEM> ::= <NAME>[<OPTIONS>]
<OPTIONS> ::= "["<OPTION>"]["["<OPTIONS>"]"]
<OPTION> ::= <PROPERTY>["="<VALUE>]
<PROPERTY> ::= "id"|"img"|"bg"|"fg"|"edit"|"sep"|"blt"|"chk"|"dis"|"bld"|"itl"|"stk"|"und"
```

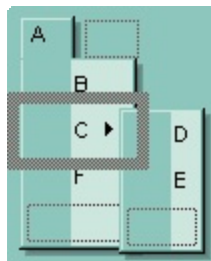
where the <NAME> is the name of the item. The <VALUE> tag is valid for the options "img", "id", "bg", "fg" and "edit", like follows:

- The <VALUE> tag for "img" option is an integer expression, that indicates the index of the icon being displayed for the item.
- The <VALUE> tag for "id" option is an integer expression, that indicates the identifier of the item.
- The <VALUE> tag for "bg" or "fg" option is a color expression, that indicates the color of the item. The <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 long expression.
- The <VALUE> tag for "edit" option is a string expression (without " characters), that indicates the caption of the edit inside the edit control.
- If the "sep" option is present, the item is a separator item.
- If the "blt" option is present, the item displays a bullet.
- If the "chk" option is present, the item displays a check box.
- If the "dis" option is present, the item is disabled.
- If the "bld" option is present, the item is bolded.
- If the "itl" option is present, the item appears as italic.
- If the "stk" option is present, the item appears as strikethrough.
- If the "und" option is present, the item is underlined.

Is there any way to get this formatted string from the control's editor? Open the control in design mode, select the Properties item in its context menu, and locate the Editor page, where you find the "Invoke Internal Editor" button, click it, and the control's Editor page is opened. Build your menu. Click an item, so it becomes focused, and press the CTRL + C, a beep should be heard. Now, open a notepad editor, and press CTRL + V, you should get the formatted string for ToString method. Click the designer, so no item is focused, and press the CTRL + C key, and so the entire menu is copied to the clipboard, as ToString syntax.

For instance, the following formatted string builds the following menu:

```
A[id=10](B[id=20],C[id=30](D[id=40],E[id=50]),F[id=60])
```



property Menu.VisibleItemsCount 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 VisibleItemsCount 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 VisibleItemsCount property specifies the number of items being visible without scroll option.



property Menu.Width as Long

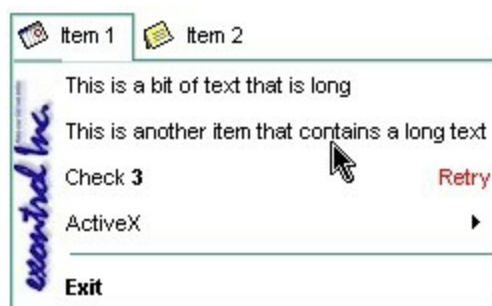
Retrieves or sets a value that indicates the width of the menu.

Type	Description
Long	A long expression that indicates the width of the menu like follows: If it is greater than 0, the value indicates the maximum width of the menu, else if it is less than 0, the absolute value indicates the exact width of the menu.

By default, the Width property is 0. If the Width property is 0, it has no effect. Use the Width property to truncate long strings. Use the Width property to specify the exact width of the menu no matter what's the length for the items. Use the [Add](#) method to add new items to the menu. Use the [Visible](#) property to specify whether an item is visible or hidden.

- The Width property has no effect if it is 0.
- The Width property indicates the maximum width of the menu, if the value is positive.
- The Width property indicates the exact width of the menu, if the value is negative.

The following screen shot shows the menu when the Width property is 0 (zero):



The following screen shot shows the menu when the Width property is 128 (positive value):



The following screen shot shows the menu when the Width property is -256 (negative value):

Item 1

Item 2

excelsior inc.

This is a bit of text that is long

This is another item that contains a long text

Check 3

ActiveX



Retry



Exit

OleEvent object

The OleEvent object holds information about an event fired by an ActiveX control hosted by a popup menu that's created using the [Add](#) method. The control fires the [OleEvent](#) event when the inside ActiveX control fires an event. The OleEvent object supports the following properties and methods:

Name	Description
CountParam	Retrieves the count of the OLE event's arguments.
ID	Retrieves a long expression that specifies the identifier of the event.
Name	Retrieves the original name of the fired event.
Param	Retrieves an OleEventParam object given either the index of the parameter, or its name.
ToString	Retrieves information about the event.

property OleEvent.CountParam as Long

Retrieves the count of the OLE event's arguments.

Type	Description
Long	A long value that indicates the count of the arguments

The [Name](#) property gets the name of the event that an inside ActiveX control fires. Use the CountParam property to count the number of parameters that the inside event carries. Use the [Param](#) property to retrieve a parameter of the inside ActiveX event.

The following VB sample prints the events that an inside ActiveX control fires:

```
Private Sub ExMenu1_OleEvent(ByVal ID As Long, ByVal Ev As EXMENUlibCtl.IOleEvent)
    With ExMenu1.Item(ID)
        Debug.Print (.Caption)
    End With
    With Ev
        Debug.Print .Name
        Dim i As Long
        For i = 0 To .CountParam - 1
            With .Param(i)
                Debug.Print .Name & " = " & .Value
            End With
        Next
    End With
End Sub
```

The following C++ sample prints the events that an inside ActiveX control fires:

```
#import <exmenu.dll>
void OnOleEventExmenu1(long ID, LPDISPATCH Ev)
{
    EXMENUlib::IOleEventPtr spEvent( Ev );
    OutputDebugString( spEvent->GetName() );
    for ( long i = 0; i < spEvent->CountParam; i++ )
    {
        EXMENUlib::IOleEventParamPtr spParam = NULL;
        spEvent->get_Param( COleVariant( i ), &spParam );
    }
}
```

```

CString strOutput;
strOutput.Format( "%s = %s", spParam->Name.operator const char *(), V2S(
&spParam->Value ) );
OutputDebugString( strOutput );
}
}

```

The C++ sample requires `#import <exmenu.dll>` to import definitions for [OleEvent](#) and [OleEvenParam](#) objects. It generates the EXMENULib namespace that includes all objects for the ExMenu component.

The V2S function converts a VARIANT value to a string expression:

```

static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}

```

The following VB.NET sample prints the events that an inside ActiveX control fires:

```

Private Sub AxExMenu1_OleEvent(ByVal sender As Object, ByVal e As
AxEXMENULib.IMenuEvents_OleEventEvent) Handles AxExMenu1.OleEvent
    With AxExMenu1.item(e.iD)
        Debug.Print(.Caption)
    End With
    With e.ev
        Debug.WriteLine(.Name())
        Dim i As Integer
        For i = 0 To .CountParam - 1
            With .Param(i)

```

```
Debug.WriteLine(.Name + " = " + .Value.ToString())
```

```
End With
```

```
Next
```

```
End With
```

```
End Sub
```

The following C# sample prints the events that an inside ActiveX control fires:

```
private void axExMenu1_OleEvent(object sender,
AxEXMENUMLib._IMenuEvents_OleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine(axExMenu1[e.iD].Caption);
    System.Diagnostics.Debug.WriteLine(e.ev.Name);
    for (int i = 0; i < e.ev.CountParam; i++)
    {
        EXMENUMLib.OleEventParam param = e.ev[i];
        System.Diagnostics.Debug.WriteLine( param.Name + " = " + param.Value.ToString());
    }
}
```

The following VFP sample prints the events that an inside ActiveX control fires:

```
*** ActiveX Control Event ***
LPARAMETERS id, ev

With thisform.ExMenu1.Item(ID)
    wait window nowait (.Caption)
EndWith

With Ev
    wait window nowait .Name
    local i
    For i = 0 To .CountParam - 1
        With .Param(i)
            wait window nowait .Name && + " = " + Str(.Value)
        EndWith
    Next
EndWith
```


property OleEvent.ID as Long

Retrieves a long expression that specifies the identifier of the event.

Type	Description
Long	A Long expression that defines the identifier of the OLE event.

The identifier of the event could be used to identify a specified OLE event. Use the [Name](#) property of the OLE Event to get the name of the OLE Event. Use the [ToString](#) property to display information about an OLE event. The ToString property displays the identifier of the event after the name of the event in two [] brackets. For instance, the ToString property gets the "KeyDown[-602](KeyCode/Short* = 9,Shift/Short = 0)" when TAB key is pressed, so the identifier of the KeyDown event being fired by the inside User editor is -602.

property OleEvent.Name as String

Retrieves the original name of the fired event.

Type	Description
String	A string expression that indicates the event's name

Use the [ID](#) property to specify a specified even by its identifier. Use the [ToString](#) property to display information about fired event such us name, parameters, types and values. Use the [CountParam](#) property to count the parameters of an OLE event. Use the [Param](#) property to get the event's parameter. Use the [Value](#) property to specify the value of the parameter.

The following VB sample prints the events that an inside ActiveX control fires:

```
Private Sub ExMenu1_OleEvent(ByVal ID As Long, ByVal Ev As EXMENULibCtl.IOleEvent)
    With ExMenu1.Item(ID)
        Debug.Print (.Caption)
    End With
    With Ev
        Debug.Print .Name
        Dim i As Long
        For i = 0 To .CountParam - 1
            With .Param(i)
                Debug.Print .Name & " = " & .Value
            End With
        Next
    End With
End Sub
```

The following C++ sample prints the events that an inside ActiveX control fires:

```
#import <exmenu.dll>
void OnOleEventExmenu1(long ID, LPDISPATCH Ev)
{
    EXMENULib::IOleEventPtr spEvent( Ev );
    OutputDebugString( spEvent->GetName() );
    for ( long i = 0; i < spEvent->CountParam; i++ )
    {
        EXMENULib::IOleEventParamPtr spParam = NULL;
```

```

spEvent->get_Param( COleVariant( i ), &spParam );
CString strOutput;
strOutput.Format( "%s = %s", spParam->Name.operator const char *(), V2S(
&spParam->Value ) );
OutputDebugString( strOutput );
}
}

```

The C++ sample requires `#import <exmenu.dll>` to import definitions for [OleEvent](#) and [OleEvenParam](#) objects. It generates the `EXMENUMLib` namespace that includes all objects for the `ExMenu` component.

The `V2S` function converts a `VARIANT` value to a string expression:

```

static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}

```

The following `VB.NET` sample prints the events that an inside `ActiveX` control fires:

```

Private Sub AxExMenu1_OleEvent(ByVal sender As Object, ByVal e As
AxEXMENUMLib._IMenuEvents_OleEventEvent) Handles AxExMenu1.OleEvent
    With AxExMenu1.item(e.iD)
        Debug.Print(.Caption)
    End With
    With e.ev
        Debug.WriteLine(.Name())
        Dim i As Integer
        For i = 0 To .CountParam - 1

```

```

    With .Param(i)
        Debug.WriteLine(.Name + " = " + .Value.ToString())
    End With
Next
End With
End Sub

```

The following C# sample prints the events that an inside ActiveX control fires:

```

private void axExMenu1_OleEvent(object sender,
AxEXMENULib._IMenuEvents_OleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine(axExMenu1[e.iD].Caption);
    System.Diagnostics.Debug.WriteLine(e.ev.Name);
    for (int i = 0; i < e.ev.CountParam; i++)
    {
        EXMENULib.OleEventParam param = e.ev[i];
        System.Diagnostics.Debug.WriteLine( param.Name + " = " + param.Value.ToString());
    }
}

```

The following VFP sample prints the events that an inside ActiveX control fires:

```

*** ActiveX Control Event ***
LPARAMETERS id, ev

With thisform.ExMenu1.Item(ID)
    wait window nowait (.Caption)
EndWith

With Ev
    wait window nowait .Name
    local i
    For i = 0 To .CountParam - 1
        With .Param(i)
            wait window nowait .Name && + " = " + Str(.Value)
        EndWith
    Next
EndWith

```


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 variant expression that indicates a argument's index or an argument's name.
OleEventParam	An OleEventParam object that contains the name and the value for the argument.

The [Name](#) property gets the name of the event that an inside ActiveX control fires. Use the [CountParam](#) property to count the number of parameters that the inside event carries. Use the Param property to retrieve a parameter of the inside ActiveX event.

The following VB sample prints the events that an inside ActiveX control fires:

```
Private Sub ExMenu1_OleEvent(ByVal ID As Long, ByVal Ev As EXMENULibCtl.IOleEvent)
    With ExMenu1.Item(ID)
        Debug.Print (.Caption)
    End With
    With Ev
        Debug.Print .Name
        Dim i As Long
        For i = 0 To .CountParam - 1
            With .Param(i)
                Debug.Print .Name & " = " & .Value
            End With
        Next
    End With
End Sub
```

The following C++ sample prints the events that an inside ActiveX control fires:

```
#import <exmenu.dll>
void OnOleEventExmenu1(long ID, LPDISPATCH Ev)
{
    EXMENULib::IOleEventPtr spEvent( Ev );
    OutputDebugString( spEvent->GetName() );
    for ( long i = 0; i < spEvent->CountParam; i++ )
    {
```

```

EXMENUMLib::IOleEventParamPtr spParam = NULL;
spEvent->get_Param( COleVariant( i ), &spParam );
CString strOutput;
strOutput.Format( "%s = %s", spParam->Name.operator const char *(), V2S(
&spParam->Value ) );
OutputDebugString( strOutput );
}
}

```

The C++ sample requires `#import <exmenu.dll>` to import definitions for [OleEvent](#) and [OleEvenParam](#) objects. It generates the EXMENUMLib namespace that includes all objects for the ExMenu component.

The V2S function converts a VARIANT value to a string expression:

```

static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}

```

The following VB.NET sample prints the events that an inside ActiveX control fires:

```

Private Sub AxExMenu1_OleEvent(ByVal sender As Object, ByVal e As
AxEXMENUMLib._IMenuEvents_OleEventEvent) Handles AxExMenu1.OleEvent
    With AxExMenu1.item(e.iD)
        Debug.Print(.Caption)
    End With
    With e.ev
        Debug.WriteLine(.Name())
    End With
    Dim i As Integer

```

```

For i = 0 To .CountParam - 1
    With .Param(i)
        Debug.WriteLine(.Name + " = " + .Value.ToString())
    End With
Next
End With
End Sub

```

The following C# sample prints the events that an inside ActiveX control fires:

```

private void axExMenu1_OleEvent(object sender,
AxEXMENUMLib._IMenuEvents_OleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine(axExMenu1[e.iD].Caption);
    System.Diagnostics.Debug.WriteLine(e.ev.Name);
    for (int i = 0; i < e.ev.CountParam; i++)
    {
        EXMENUMLib.OleEventParam param = e.ev[i];
        System.Diagnostics.Debug.WriteLine( param.Name + " = " + param.Value.ToString());
    }
}

```

The following VFP sample prints the events that an inside ActiveX control fires:

```

*** ActiveX Control Event ***
LPARAMETERS id, ev

With thisform.ExMenu1.Item(ID)
    wait window nowait (.Caption)
EndWith

With Ev
    wait window nowait .Name
    local i
    For i = 0 To .CountParam - 1
        With .Param(i)
            wait window nowait .Name && + " = " + Str(.Value)
        EndWith
    Next

```


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. The control fires the [OleEvent](#) event when the inside ActiveX control fires an event. The OleEventParam object supports the following properties and methods:

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.

The Name property gets the name of the parameter. The [Value](#) property specifies the value of the parameter.

The following VB sample prints the events that an inside ActiveX control fires:

```
Private Sub ExMenu1_OleEvent(ByVal ID As Long, ByVal Ev As EXMENUlibCtl.IOleEvent)
    With ExMenu1.Item(ID)
        Debug.Print (.Caption)
    End With
    With Ev
        Debug.Print .Name
        Dim i As Long
        For i = 0 To .CountParam - 1
            With .Param(i)
                Debug.Print .Name & " = " & .Value
            End With
        Next
    End With
End Sub
```

The following C++ sample prints the events that an inside ActiveX control fires:

```
#import <exmenu.dll>
void OnOleEventExmenu1(long ID, LPDISPATCH Ev)
{
    EXMENUlib::IOleEventPtr spEvent( Ev );
    OutputDebugString( spEvent->GetName() );
    for ( long i = 0; i < spEvent->CountParam; i++ )
    {
        EXMENUlib::IOleEventParamPtr spParam = NULL;
        spEvent->get_Param( COleVariant( i ), &spParam );
    }
}
```

```

CString strOutput;
strOutput.Format( "%s = %s", spParam->Name.operator const char *(), V2S(
&spParam->Value ) );
OutputDebugString( strOutput );
}
}

```

The C++ sample requires `#import <exmenu.dll>` to import definitions for [OleEvent](#) and [OleEvenParam](#) objects. It generates the EXMENULib namespace that includes all objects for the ExMenu component.

The V2S function converts a VARIANT value to a string expression:

```

static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}

```

The following VB.NET sample prints the events that an inside ActiveX control fires:

```

Private Sub AxExMenu1_OleEvent(ByVal sender As Object, ByVal e As
AxEXMENULib.IMenuEvents_OleEventEvent) Handles AxExMenu1.OleEvent
    With AxExMenu1.item(e.iD)
        Debug.Print(.Caption)
    End With
    With e.ev
        Debug.WriteLine(.Name())
        Dim i As Integer
        For i = 0 To .CountParam - 1
            With .Param(i)

```

```
Debug.WriteLine(.Name + " = " + .Value.ToString())
```

```
End With
```

```
Next
```

```
End With
```

```
End Sub
```

The following C# sample prints the events that an inside ActiveX control fires:

```
private void axExMenu1_OleEvent(object sender,
AxEXMENULib._IMenuEvents_OleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine(axExMenu1[e.iD].Caption);
    System.Diagnostics.Debug.WriteLine(e.ev.Name);
    for (int i = 0; i < e.ev.CountParam; i++)
    {
        EXMENULib.OleEventParam param = e.ev[i];
        System.Diagnostics.Debug.WriteLine( param.Name + " = " + param.Value.ToString());
    }
}
```

The following VFP sample prints the events that an inside ActiveX control fires:

```
*** ActiveX Control Event ***
LPARAMETERS id, ev

With thisform.ExMenu1.Item(ID)
    wait window nowait (.Caption)
EndWith

With Ev
    wait window nowait .Name
    local i
    For i = 0 To .CountParam - 1
        With .Param(i)
            wait window nowait .Name && + " = " + Str(.Value)
        EndWith
    Next
EndWith
```


property OleEventParam.Value as Variant

Retrieves the value of the event's parameter.

Type	Description
Variant	A variant value that indicates the value of the event's parameter.

Use the [ID](#) property to specify a specified even by its identifier. Use the [ToString](#) property to display information about fired event such us name, parameters, types and values. Use the [CountParam](#) property to count the parameters of an OLE event. Use the [Param](#) property to get the event's parameter. Use the [Value](#) property to specify the value of the parameter.

The following VB sample prints the events that an inside ActiveX control fires:

```
Private Sub ExMenu1_OleEvent(ByVal ID As Long, ByVal Ev As EXMENUlibCtl.IOleEvent)
    With ExMenu1.Item(ID)
        Debug.Print (.Caption)
    End With
    With Ev
        Debug.Print .Name
        Dim i As Long
        For i = 0 To .CountParam - 1
            With .Param(i)
                Debug.Print .Name & " = " & .Value
            End With
        Next
    End With
End Sub
```

The following C++ sample prints the events that an inside ActiveX control fires:

```
#import <exmenu.dll>
void OnOleEventExmenu1(long ID, LPDISPATCH Ev)
{
    EXMENUlib::IOleEventPtr spEvent( Ev );
    OutputDebugString( spEvent->GetName() );
    for ( long i = 0; i < spEvent->CountParam; i++ )
    {
        EXMENUlib::IOleEventParamPtr spParam = NULL;
```

```

spEvent->get_Param( COleVariant( i ), &spParam );
CString strOutput;
strOutput.Format( "%s = %s", spParam->Name.operator const char *(), V2S(
&spParam->Value ) );
OutputDebugString( strOutput );
}
}

```

The C++ sample requires `#import <exmenu.dll>` to import definitions for [OleEvent](#) and [OleEvenParam](#) objects. It generates the `EXMENUMLib` namespace that includes all objects for the ExMenu component.

The `V2S` function converts a `VARIANT` value to a string expression:

```

static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}

```

The following VB.NET sample prints the events that an inside ActiveX control fires:

```

Private Sub AxExMenu1_OleEvent(ByVal sender As Object, ByVal e As
AxEXMENUMLib._IMenuEvents_OleEventEvent) Handles AxExMenu1.OleEvent
    With AxExMenu1.item(e.iD)
        Debug.Print(.Caption)
    End With
    With e.ev
        Debug.WriteLine(.Name())
        Dim i As Integer
        For i = 0 To .CountParam - 1

```

```

With .Param(i)
    Debug.WriteLine(.Name + " = " + .Value.ToString())
End With
Next
End With
End Sub

```

The following C# sample prints the events that an inside ActiveX control fires:

```

private void axExMenu1_OleEvent(object sender,
AxEXMENULib._IMenuEvents_OleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine(axExMenu1[e.iD].Caption);
    System.Diagnostics.Debug.WriteLine(e.ev.Name);
    for (int i = 0; i < e.ev.CountParam; i++)
    {
        EXMENULib.OleEventParam param = e.ev[i];
        System.Diagnostics.Debug.WriteLine( param.Name + " = " + param.Value.ToString());
    }
}

```

The following VFP sample prints the events that an inside ActiveX control fires:

```

*** ActiveX Control Event ***
LPARAMETERS id, ev

With thisform.ExMenu1.Item(ID)
    wait window nowait (.Caption)
EndWith

With Ev
    wait window nowait .Name
    local i
    For i = 0 To .CountParam - 1
        With .Param(i)
            wait window nowait .Name && + " = " + Str(.Value)
        EndWith
    Next
EndWith

```


ExMenu events

The ExMenu component supports the following events:

Name	Description
Click	Fired when the user clicks an item.
ClosePopup	Fired when a sub menu is about to be closed.
EditChange	Fired when the caption of the item's edit was altered
Highlight	Fired when an item is highlighted.
OleEvent	Occurs when an ActiveX inside control fires an event.
OpenPopup	Fired when a sub menu is about to be opened.
RClick	Fired when the user right clicks an item.
Select	Occurs when an item is selected by clicking or by pressing RETURN key.

event Click (ID as Long)

Fired when the user clicks an item.

Type	Description
ID as Long	A long expression that specifies the identifier of the item being clicked.

The Click event is fired when the user clicks the item. Use the [Select](#) event to notify your application that the user selects an item. Use the [RClick](#) event to notify your application that user right clicked an item. Use the [Item](#) property to access an item giving its identifier. Use the [Caption](#) property to specify the caption of the item. Use the [SelectOn](#) property to specify whether the control selects an item if the user presses or releases the mouse button.

Syntax for Click event, **/NET** version, on:

```
C# private void Click(object sender,int ID)
{
}
```

```
VB Private Sub Click(ByVal sender As System.Object,ByVal ID As Integer) Handles Click
End Sub
```

Syntax for Click event, **/COM** version, on:

```
C# private void ClickEvent(object sender, AxEXMENULib._IMenuEvents_ClickEvent e)
{
}
```

```
C++ void OnClick(long ID)
{
}
```

```
C++ Builder void __fastcall Click(TObject *Sender,long ID)
{
}
```

```
Delphi procedure Click(ASender: TObject; ID : Integer);
begin
end;
```

```
Delphi 8  
(.NET  
only) procedure ClickEvent(sender: System.Object; e:  
AxEXMENUMLib._IMenuEvents_ClickEvent);  
begin  
end;
```

```
Powe... begin event Click(long ID)  
end event Click
```

```
VB.NET Private Sub ClickEvent(ByVal sender As System.Object, ByVal e As  
AxEXMENUMLib._IMenuEvents_ClickEvent) Handles ClickEvent  
End Sub
```

```
VB6 Private Sub Click(ID As Long)  
End Sub
```

```
VBA Private Sub Click(ByVal ID As Long)  
End Sub
```

```
VFP LPARAMETERS ID
```

```
Xbas... PROCEDURE OnClick(oExMenu,ID)  
RETURN
```

Syntax for Click event, **ICOM** version (others), on:

```
Java... <SCRIPT EVENT="Click(ID)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBSc... <SCRIPT LANGUAGE="VBScript">  
Function Click(ID)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComClick Integer IID  
Forward Send OnComClick IID  
End_Procedure
```

Visual
Objects

```
METHOD OCX_Click(ID) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_Click(int _ID)  
{  
}
```

XBasic

```
function Click as v (ID as N)  
end function
```

dBASE

```
function nativeObject_Click(ID)  
return
```

The following VB sample displays the caption of the item being clicked:

```
Private Sub ExMenu1_Click(ID As Long)  
    Debug.Print ExMenu1.Item(ID).Caption  
End Sub
```

The following C++ sample displays the caption of the item being clicked:

```
#include "Item.h"  
void OnClickExmenu1(long ID)  
{  
    OutputDebugString( m_menu.GetItem( COleVariant(ID) ).GetCaption() );  
}
```

The following VB.NET sample displays the caption of the item being clicked:

```
Private Sub AxExMenu1_ClickEvent(ByVal sender As Object, ByVal e As  
AxEXMENULib._IMenuEvents_ClickEvent) Handles AxExMenu1.ClickEvent  
    Debug.WriteLine(AxExMenu1.item(e.iD).Caption)  
End Sub
```

The following C# sample displays the caption of the item being clicked:

```
private void axExMenu1_ClickEvent(object sender,  
AxEXMENULib._IMenuEvents_ClickEvent e)  
{
```

```
System.Diagnostics.Debug.WriteLine(axExMenu1[e.iD].Caption);  
}
```

The following VFP sample displays the caption of the item being clicked:

```
*** ActiveX Control Event ***  
LPARAMETERS id  
  
with thisform.ExMenu1  
    wait window nowait .Item(id).Caption  
endwith
```

event ClosePopup (ID as Long)

Fired when a sub menu is about to be closed.

Type	Description
ID as Long	A long expression that specifies the identifier of the item being closed.

The ClosePopup event occurs when a popup menu is closed. The [OpenPopup](#) event is fired when user opens a popup menu. Use the [Select](#) event to notify your application that an item is selected. Use the [Popup](#) property to specify whether an item contains a sub menu. Use the [SubMenu](#) property to retrieve the sub menu. Use the [Item](#) property to access an item giving its identifier.

Syntax for ClosePopup event, **/NET** version, on:

```
C# private void ClosePopup(object sender,int ID)
{
}
```

```
VB Private Sub ClosePopup(ByVal sender As System.Object,ByVal ID As Integer)
Handles ClosePopup
End Sub
```

Syntax for ClosePopup event, **/COM** version, on:

```
C# private void ClosePopup(object sender,
AxEXMENULib._IMenuEvents_ClosePopupEvent e)
{
}
```

```
C++ void OnClosePopup(long ID)
{
}
```

```
C++ Builder void __fastcall ClosePopup(TObject *Sender,long ID)
{
}
```

```
Delphi procedure ClosePopup(ASender: TObject; ID : Integer);
```

```
begin  
end;
```

Delphi 8
(.NET
only)

```
procedure ClosePopup(sender: System.Object; e:  
AxEXMENULib._IMenuEvents_ClosePopupEvent);  
begin  
end;
```

Power...

```
begin event ClosePopup(long ID)  
end event ClosePopup
```

VB.NET

```
Private Sub ClosePopup(ByVal sender As System.Object, ByVal e As  
AxEXMENULib._IMenuEvents_ClosePopupEvent) Handles ClosePopup  
End Sub
```

VB6

```
Private Sub ClosePopup(ByVal ID As Long)  
End Sub
```

VBA

```
Private Sub ClosePopup(ByVal ID As Long)  
End Sub
```

VFP

```
LPARAMETERS ID
```

Xbas...

```
PROCEDURE OnClosePopup(oExMenu,ID)  
RETURN
```

Syntax for ClosePopup event, **ICOM** version (others), on:

Java...

```
<SCRIPT EVENT="ClosePopup(ID)" LANGUAGE="JScript" >  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript" >  
Function ClosePopup(ID)  
End Function  
</SCRIPT>
```

Visual
Data...

```
Procedure OnComClosePopup Integer IID  
    Forward Send OnComClosePopup IID  
End_Procedure
```

Visual
Objects

```
METHOD OCX_ClosePopup(ID) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_ClosePopup(int _ID)  
{  
}
```

XBasic

```
function ClosePopup as v (ID as N)  
end function
```

dBASE

```
function nativeObject_ClosePopup(ID)  
return
```

In VFP9, if using the `_SCREEN.AddObject` on modal forms, the `OpenPopup` may not be fired, instead you can use the `BINDEVENT` to collect the `WM_COMMAND` message. The `IPParam` of the `WM_COMMAND` message is 3 for the `ClosePopup` event.

The following VB sample displays the caption of the item whose sub menu is closed:

```
Private Sub ExMenu1_ClosePopup(ByVal ID As Long)  
    Debug.Print ExMenu1.Item(ID).Caption  
End Sub
```

The following C++ sample displays the caption of the item whose sub menu is close:

```
#include "Item.h"  
void OnClosePopupExmenu1(long ID)  
{  
    OutputDebugString( m_menu.GetItem( COleVariant(ID) ).GetCaption() );  
}
```

The following VB.NET sample displays the caption of the item whose sub menu is close:

```
Private Sub AxExMenu1_ClosePopup(ByVal sender As Object, ByVal e As  
AxEXMENULib._IMenuEvents_ClosePopupEvent) Handles AxExMenu1.ClosePopup  
    Debug.WriteLine(AxExMenu1.item(e.iD).Caption)  
End Sub
```

The following C# sample displays the caption of the item whose sub menu is close:

```
private void axExMenu1_ClosePopup(object sender,  
AxEXMENULib._IMenuEvents_ClosePopupEvent e)  
{  
    System.Diagnostics.Debug.WriteLine(axExMenu1[e.iD].Caption);  
}
```

The following VFP sample displays the caption of the item whose sub menu is close:

```
*** ActiveX Control Event ***  
LPARAMETERS id  
  
with thisform.ExMenu1  
    wait window nowait .Item(id).Caption  
endwith
```

event EditChange (ID as Long)

Fired when the caption of the item's edit was altered

Type	Description
ID as Long	A long expression that indicates the item's identifier.

The EditChange event is fired when the user alters the item's edit caption. The EditChange event is fired only if the [AllowEdit](#) property is True. Use the [EditCaption](#) property to get the edit's caption for an item. Use the [Item](#) property to get an menu's item given its identifier.

Syntax for EditChange event, **/NET** version, on:

```
C# private void EditChange(object sender,int ID)
{
}
```

```
VB Private Sub EditChange(ByVal sender As System.Object,ByVal ID As Integer)
Handles EditChange
End Sub
```

Syntax for EditChange event, **/COM** version, on:

```
C# private void EditChange(object sender,
AxEXMENULib._IMenuEvents_EditChangeEvent e)
{
}
```

```
C++ void OnEditChange(long ID)
{
}
```

```
C++ Builder void __fastcall EditChange(TObject *Sender,long ID)
{
}
```

```
Delphi procedure EditChange(ASender: TObject; ID : Integer);
begin
end;
```

Delphi 8
(.NET
only)

```
procedure EditChange(sender: System.Object; e:  
AxEXMENULib._IMenuEvents_EditChangeEvent);  
begin  
end;
```

Power...

```
begin event EditChange(long ID)  
end event EditChange
```

VB.NET

```
Private Sub EditChange(ByVal sender As System.Object, ByVal e As  
AxEXMENULib._IMenuEvents_EditChangeEvent) Handles EditChange  
End Sub
```

VB6

```
Private Sub EditChange(ByVal ID As Long)  
End Sub
```

VBA

```
Private Sub EditChange(ByVal ID As Long)  
End Sub
```

VFP

```
LPARAMETERS ID
```

Xbas...

```
PROCEDURE OnEditChange(oExMenu,ID)  
RETURN
```

Syntax for EditChange event, **ICOM** version (others), on:

Java...

```
<SCRIPT EVENT="EditChange(ID)" LANGUAGE="JScript">  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function EditChange(ID)  
End Function  
</SCRIPT>
```

Visual
Data...

```
Procedure OnComEditChange Integer IID
```

```
Forward Send OnComEditChange IID
End_Procedure
```

Visual
Objects

```
METHOD OCX_EditChange(ID) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_EditChange(int _ID)
{
}
```

XBasic

```
function EditChange as v (ID as N)
end function
```

dBASE

```
function nativeObject_EditChange(ID)
return
```

The following VB sample displays the caption of the edit control when the user alters it:

```
Private Sub ExMenu1_EditChange(ByVal ID As Long)
    With ExMenu1.Item(ID)
        Debug.Print .Caption & " " & .EditCaption
    End With
End Sub
```

The following C++ sample displays the caption of the edit control when user alters it:

```
#include "Item.h"
void OnEditChangeExmenu1(long ID)
{
    CItem item = m_menu.GetItem( COleVariant(ID) );
    OutputDebugString( item.GetCaption() );
    OutputDebugString( item.GetEditCaption() );
}
```

The following VB.NET sample displays the caption of the edit control when user alters it:

```
Private Sub AxExMenu1_EditChange(ByVal sender As Object, ByVal e As
AxEXMENU.Lib._IMenuEvents_EditChangeEvent) Handles AxExMenu1.EditChange
```

```
With AxExMenu1.item(e.iD)
    Debug.WriteLine(.Caption & " " & .EditCaption)
End With
End Sub
```

The following C# sample displays the caption of the edit control when user alters it:

```
private void axExMenu1_EditChange(object sender,
AxEXMENULib._IMenuEvents_EditChangeEvent e)
{
    EXMENULib.item item = axExMenu1[e.iD];
    System.Diagnostics.Debug.WriteLine(item.Caption + " " + item.EditCaption );
}
```

The following VFP sample displays the caption of the edit control when user alters it:

```
*** ActiveX Control Event ***
LPARAMETERS id

with thisform.ExMenu1.Item(id)
    wait window nowait .Caption + " " + .EditChange
endwith
```

event Highlight (ID as Long)

Fired when an item is highlighted.

Type	Description
ID as Long	A long expression that indicates the identifier of the item being highlighted.

The Highlight event occurs when the cursor hovers the item. Use the [Item](#) property to access an item giving its identifier. Use the [Select](#) event property to notify your application that a new items is selected. Use the [Caption](#) property to specify the caption of the item.

Syntax for Highlight event, **/NET** version, on:

```
C# private void Highlight(object sender,int ID)
{
}
```

```
VB Private Sub Highlight(ByVal sender As System.Object,ByVal ID As Integer) Handles
Highlight
End Sub
```

Syntax for Highlight event, **/COM** version, on:

```
C# private void Highlight(object sender, AxEXMENUlib._IMenuEvents_HighlightEvent
e)
{
}
```

```
C++ void OnHighlight(long ID)
{
}
```

```
C++ Builder void __fastcall Highlight(TObject *Sender,long ID)
{
}
```

```
Delphi procedure Highlight(ASender: TObject; ID : Integer);
begin
end;
```

```
Delphi 8  
(.NET  
only) procedure Highlight(sender: System.Object; e:  
AxEXMENULib._IMenuEvents_HighlightEvent);  
begin  
end;
```

```
Powe... begin event Highlight(long ID)  
end event Highlight
```

```
VB.NET Private Sub Highlight(ByVal sender As System.Object, ByVal e As  
AxEXMENULib._IMenuEvents_HighlightEvent) Handles Highlight  
End Sub
```

```
VB6 Private Sub Highlight(ByVal ID As Long)  
End Sub
```

```
VBA Private Sub Highlight(ByVal ID As Long)  
End Sub
```

```
VFP LPARAMETERS ID
```

```
Xbas... PROCEDURE OnHighlight(oExMenu,ID)  
RETURN
```

Syntax for Highlight event, **/COM** version (others), on:

```
Java... <SCRIPT EVENT="Highlight(ID)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBSc... <SCRIPT LANGUAGE="VBScript">  
Function Highlight(ID)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComHighlight Integer IID  
Forward Send OnComHighlight IID  
End_Procedure
```

```
METHOD OCX_Highlight(ID) CLASS MainDialog  
RETURN NIL
```

```
X++  
void onEvent_Highlight(int _ID)  
{  
}
```

```
XBasic  
function Highlight as v (ID as N)  
end function
```

```
dBASE  
function nativeObject_Highlight(ID)  
return
```

The following VB sample displays the caption of the highlighted item:

```
Private Sub ExMenu1_Highlight(ByVal ID As Long)  
    Debug.Print ExMenu1.Item(ID).Caption  
End Sub
```

The following C++ sample displays the caption of the highlighted item:

```
#include "Item.h"  
void OnHighlightExmenu1(long ID)  
{  
    CItem item = m_menu.GetItem( COleVariant(ID) );  
    OutputDebugString( item.GetCaption() );  
}
```

The following VB.NET sample displays the caption of the highlighted item:

```
Private Sub AxExMenu1_Highlight(ByVal sender As Object, ByVal e As  
AxEXMENULib._IMenuEvents_HighlightEvent) Handles AxExMenu1.Highlight  
    Debug.WriteLine(AxExMenu1.item(e.iID).Caption)  
End Sub
```

The following C# sample displays the caption of the highlighted item:

```
private void axExMenu1_Highlight(object sender,  
AxEXMENUMLib._IMenuEvents_HighlightEvent e)  
{  
    System.Diagnostics.Debug.WriteLine(axExMenu1[e.iD].Caption);  
}
```

The following VFP sample displays the caption of the highlighted item:

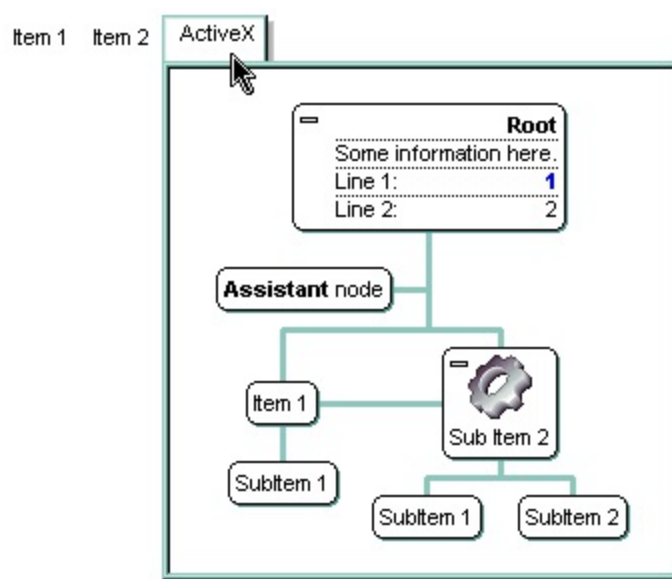
```
*** ActiveX Control Event ***  
LPARAMETERS id  
  
with thisform.ExMenu1.Item(id)  
    wait window nowait .Caption  
endwith
```

event OleEvent (ID as Long, Ev as OleEvent)

Occurs when an ActiveX inside control fires an event.

Type	Description
ID as Long	A long expression that indicates the identifier of the item that hosts an ActiveX control.
Ev as OleEvent	An OleEvent object that contains information about the event.

Use the [Item](#) property to access the menu's item by its identifier. Use the [Add](#) method to insert a popup menu that hosts an ActiveX control. Use the [ControllID](#) property to specify the control's identifier. Use the [Create](#) method to creates an ActiveX control inside a popup menu. The [CloseOn](#) property indicates when the popup menu is closed. Use the [Refresh](#) method to refresh the control after adding items to the menu. Use the [Caption](#) property to specify the caption of the item.



Syntax for OleEvent event, **/NET** version, on:

```
C# private void OleEvent(object sender,int ID,excontrol.EXMENUlib.OleEvent Ev)
{
}
```

```
VB Private Sub OleEvent(ByVal sender As System.Object,ByVal ID As Integer,ByVal Ev
As excontrol.EXMENUlib.OleEvent) Handles OleEvent
End Sub
```

Syntax for OleEvent event, **/COM** version, on:

C#

```
private void OleEvent(object sender, AxEXMENUMLib._IMenuEvents_OleEventEvent e)
{
}
```

C++

```
void OnOleEvent(long ID,LPDISPATCH Ev)
{
}
```

C++
Builder

```
void __fastcall OleEvent(TObject *Sender,long ID,Exmenulib_tlb::IOleEvent *Ev)
{
}
```

Delphi

```
procedure OleEvent(ASender: TObject; ID : Integer;Ev : IOleEvent);
begin
end;
```

Delphi 8
(.NET
only)

```
procedure OleEvent(sender: System.Object; e:
AxEXMENUMLib._IMenuEvents_OleEventEvent);
begin
end;
```

Powe...

```
begin event OleEvent(long ID,oleobject Ev)
end event OleEvent
```

VB.NET

```
Private Sub OleEvent(ByVal sender As System.Object, ByVal e As
AxEXMENUMLib._IMenuEvents_OleEventEvent) Handles OleEvent
End Sub
```

VB6

```
Private Sub OleEvent(ByVal ID As Long,ByVal Ev As EXMENUMLibCtl.IOleEvent)
End Sub
```

VBA

```
Private Sub OleEvent(ByVal ID As Long,ByVal Ev As Object)
End Sub
```

VFP

```
LPARAMETERS ID,Ev
```

Xbas...

```
PROCEDURE OnOleEvent(oExMenu, ID, Ev)
RETURN
```

Syntax for OleEvent event, **ICOM** version (others), on:

Java...

```
<SCRIPT EVENT="OleEvent(ID,Ev)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">
Function OleEvent(ID,Ev)
End Function
</SCRIPT>
```

Visual
Data...

```
Procedure OnComOleEvent Integer IID Variant IEv
Forward Send OnComOleEvent IID IEv
End_Procedure
```

Visual
Objects

```
METHOD OCX_OleEvent(ID,Ev) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_OleEvent(int _ID, COM _Ev)
{
}
```

XBasic

```
function OleEvent as v (ID as N, Ev as OLE::Exontrol.ExMenu.1::IOleEvent)
end function
```

dBASE

```
function nativeObject_OleEvent(ID,Ev)
return
```

The following VB sample adds a MSCAL.Calendar control, and prints the event that inside ActiveX control fires:

```
Private Sub ExMenu1_OleEvent(ByVal ID As Long, ByVal Ev As EXMENUlibCtl.IOleEvent)
    Debug.Print Ev.Name
End Sub
```

```

Private Sub Form_Load()
  With ExMenu1
    With .Items.Add("Date", ItemTypeEnum.SubControl, 0).SubControl
      .CloseOn = exLButtonUp
      .ControlID = "MSCal.Calendar"
      .Create
    End With
    .Refresh
  End With
End Sub

```

The following VB sample prints the events that an inside ActiveX control fires:

```

Private Sub ExMenu1_OleEvent(ByVal ID As Long, ByVal Ev As EXMENUlibCtl.IOleEvent)
  With ExMenu1.Item(ID)
    Debug.Print (.Caption)
  End With
  With Ev
    Debug.Print .Name
    Dim i As Long
    For i = 0 To .CountParam - 1
      With .Param(i)
        Debug.Print .Name & " = " & .Value
      End With
    Next
  End With
End Sub

```

The following C++ sample prints the events that an inside ActiveX control fires:

```

#import <exmenu.dll>
void OnOleEventExmenu1(long ID, LPDISPATCH Ev)
{
  EXMENUlib::IOleEventPtr spEvent( Ev );
  OutputDebugString( spEvent->GetName() );
  for ( long i = 0; i < spEvent->CountParam; i++ )
  {

```

```

EXMENUMLib::IOleEventParamPtr spParam = NULL;
spEvent->get_Param( COleVariant( i ), &spParam );
CString strOutput;
strOutput.Format( "%s = %s", spParam->Name.operator const char *(), V2S(
&spParam->Value ) );
OutputDebugString( strOutput );
}
}

```

The C++ sample requires `#import <exmenu.dll>` to import definitions for [OleEvent](#) and [OleEvenParam](#) objects. It generates the EXMENUMLib namespace that includes all objects for the ExMenu component.

The V2S function converts a VARIANT value to a string expression:

```

static CString V2S( VARIANT* pv, LPCTSTR szDefault = _T("") )
{
    if ( pv )
    {
        if ( pv->vt == VT_ERROR )
            return szDefault;

        COleVariant vt;
        vt.ChangeType( VT_BSTR, pv );
        return V_BSTR( &vt );
    }
    return szDefault;
}

```

The following VB.NET sample prints the events that an inside ActiveX control fires:

```

Private Sub AxExMenu1_OleEvent(ByVal sender As Object, ByVal e As
AxEXMENUMLib._IMenuEvents_OleEventEvent) Handles AxExMenu1.OleEvent
    With AxExMenu1.item(e.iD)
        Debug.Print(.Caption)
    End With
    With e.ev
        Debug.WriteLine(.Name())
    End With
    Dim i As Integer

```

```

For i = 0 To .CountParam - 1
    With .Param(i)
        Debug.WriteLine(.Name + " = " + .Value.ToString())
    End With
Next
End With
End Sub

```

The following C# sample prints the events that an inside ActiveX control fires:

```

private void axExMenu1_OleEvent(object sender,
AxEXMENUMLib._IMenuEvents_OleEventEvent e)
{
    System.Diagnostics.Debug.WriteLine(axExMenu1[e.iD].Caption);
    System.Diagnostics.Debug.WriteLine(e.ev.Name);
    for (int i = 0; i < e.ev.CountParam; i++)
    {
        EXMENUMLib.OleEventParam param = e.ev[i];
        System.Diagnostics.Debug.WriteLine( param.Name + " = " + param.Value.ToString());
    }
}

```

The following VFP sample prints the events that an inside ActiveX control fires:

```

*** ActiveX Control Event ***
LPARAMETERS id, ev

With thisform.ExMenu1.Item(ID)
    wait window nowait (.Caption)
EndWith

With Ev
    wait window nowait .Name
    local i
    For i = 0 To .CountParam - 1
        With .Param(i)
            wait window nowait .Name && + " = " + Str(.Value)
        EndWith
    Next

```


event OpenPopup (ID as Long)

Fired when a sub menu is about to be opened.

Type	Description
ID as Long	A long expression that specifies the identifier of the item being opened.

The OpenPopup event occurs when user opens a popup menu. The [ClosePopup](#) event occurs when a popup menu is closed. Use the [Select](#) event to notify your application that an item is selected. Use the [Popup](#) property to specify whether an item contains a sub menu. Use the [SubMenu](#) property to retrieve the sub menu. Use the [Item](#) property to access an item giving its identifier.

In VFP9, if using the `_SCREEN.AddObject` on modal forms, the OpenPopup may not be fired, instead you can use the BINDEVENT to collect the WM_COMMAND message. The IParam of the WM_COMMAND message is 2 for the OpenPopup event.

Syntax for OpenPopup event, **/NET** version, on:

```
C# private void OpenPopup(object sender,int ID)
{
}
```

```
VB Private Sub OpenPopup(ByVal sender As System.Object,ByVal ID As Integer)
Handles OpenPopup
End Sub
```

Syntax for OpenPopup event, **/COM** version, on:

```
C# private void OpenPopup(object sender,
AxEXMENULib._IMenuEvents_OpenPopupEvent e)
{
}
```

```
C++ void OnOpenPopup(long ID)
{
}
```

```
C++ Builder void __fastcall OpenPopup(TObject *Sender,long ID)
{
```

```
}
```

```
Delphi procedure OpenPopup(ASender: TObject; ID : Integer);  
begin  
end;
```

```
Delphi 8 (.NET only) procedure OpenPopup(sender: System.Object; e:  
AxEXMENUMLib._IMenuEvents_OpenPopupEvent);  
begin  
end;
```

```
Power... begin event OpenPopup(long ID)  
end event OpenPopup
```

```
VB.NET Private Sub OpenPopup(ByVal sender As System.Object, ByVal e As  
AxEXMENUMLib._IMenuEvents_OpenPopupEvent) Handles OpenPopup  
End Sub
```

```
VB6 Private Sub OpenPopup(ByVal ID As Long)  
End Sub
```

```
VBA Private Sub OpenPopup(ByVal ID As Long)  
End Sub
```

```
VFP LPARAMETERS ID
```

```
Xbas... PROCEDURE OnOpenPopup(oExMenu,ID)  
RETURN
```

Syntax for OpenPopup event, **ICOM** version (others), on:

```
Java... <SCRIPT EVENT="OpenPopup(ID)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBSc... <SCRIPT LANGUAGE="VBScript">  
Function OpenPopup(ID)  
End Function
```

```
</SCRIPT>
```

Visual
Data...

```
Procedure OnComOpenPopup Integer IID  
    Forward Send OnComOpenPopup IID  
End_Procedure
```

Visual
Objects

```
METHOD OCX_OpenPopup(ID) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_OpenPopup(int _ID)  
{  
}
```

XBasic

```
function OpenPopup as v (ID as N)  
end function
```

dBASE

```
function nativeObject_OpenPopup(ID)  
return
```

The following VB sample displays the caption of the item whose submenu is opened:

```
Private Sub ExMenu1_OpenPopup(ByVal ID As Long)  
    Debug.Print ExMenu1.Item(ID).Caption  
End Sub
```

The following C++ sample displays the caption of the item whose submenu is opened:

```
#include "Item.h"  
void OnOpenPopupExmenu1(long ID)  
{  
    OutputDebugString( m_menu.GetItem( COleVariant(ID) ).GetCaption() );  
}
```

The following VB.NET sample displays the caption of the item whose submenu is opened:

```
Private Sub AxExMenu1_OpenPopup(ByVal sender As Object, ByVal e As  
AxEXMENUlib._IMenuEvents_OpenPopupEvent) Handles AxExMenu1.OpenPopup  
    Debug.WriteLine(AxExMenu1.item(e.iD).Caption)
```

The following C# sample displays the caption of the item whose submenu is opened:

```
private void axExMenu1_OpenPopup(object sender,
AxEXMENU Lib._IMenuEvents_OpenPopupEvent e)
{
    System.Diagnostics.Debug.WriteLine(axExMenu1[e.iD].Caption);
}
```

The following VFP sample displays the caption of the item whose submenu is opened:

```
*** ActiveX Control Event ***
LPARAMETERS id

with thisform.ExMenu1
    wait window nowait .Item(id).Caption
endwith
```

event RClick (ID as Long)

Fired when an item has been clicked using the right mouse button.

Type	Description
ID as Long	A long expression that specifies the item's identifier being right clicked.

Use the RClick event to notify your application that an item was right clicked. Use the [Click](#) event to notify your application when user clicks an item. Use the [Item](#) property to retrieve the item giving its identifier. Use the [Select](#) event to notify you application that the user selects an item. Use the [Caption](#) property to specify the caption of the item.

Syntax for RClick event, **/NET** version, on:

```
C# private void RClick(object sender,int ID)
{
}
```

```
VB Private Sub RClick(ByVal sender As System.Object,ByVal ID As Integer) Handles
RClick
End Sub
```

Syntax for RClick event, **/COM** version, on:

```
C# private void RClick(object sender, AxEXMENULib._IMenuEvents_RClickEvent e)
{
}
```

```
C++ void OnRClick(long ID)
{
}
```

```
C++ Builder void __fastcall RClick(TObject *Sender,long ID)
{
}
```

```
Delphi procedure RClick(ASender: TObject; ID : Integer);
begin
end;
```

```
Delphi 8  
(.NET  
only) procedure RClick(sender: System.Object; e:  
AxEXMENULib._IMenuEvents_RClickEvent);  
begin  
end;
```

```
Powe... begin event RClick(long ID)  
end event RClick
```

```
VB.NET Private Sub RClick(ByVal sender As System.Object, ByVal e As  
AxEXMENULib._IMenuEvents_RClickEvent) Handles RClick  
End Sub
```

```
VB6 Private Sub RClick(ByVal ID As Long)  
End Sub
```

```
VBA Private Sub RClick(ByVal ID As Long)  
End Sub
```

```
VFP LPARAMETERS ID
```

```
Xbas... PROCEDURE OnRClick(oExMenu,ID)  
RETURN
```

Syntax for RClick event, **ICOM** version (others), on:

```
Java... <SCRIPT EVENT="RClick(ID)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript">  
Function RClick(ID)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComRClick Integer IID  
Forward Send OnComRClick IID  
End_Procedure
```

```
METHOD OCX_RClick(ID) CLASS MainDialog  
RETURN NIL
```

```
X++  
void onEvent_RClick(int _ID)  
{  
}
```

```
XBasic  
function RClick as v (ID as N)  
end function
```

```
dBASE  
function nativeObject_RClick(ID)  
return
```

The following VB sample displays the caption of the item being right clicked:

```
Private Sub ExMenu1_RClick(ByVal ID As Long)  
    Debug.Print ExMenu1.Item(ID).Caption  
End Sub
```

The following C++ sample displays the caption of the item being right clicked:

```
#include "Item.h"  
void OnRClickExmenu1(long ID)  
{  
    OutputDebugString( m_menu.GetItem( COleVariant(ID) ).GetCaption() );  
}
```

The following VB.NET sample displays the caption of the item being right clicked:

```
Private Sub AxExMenu1_RClick(ByVal sender As Object, ByVal e As  
AxEXMENU.Lib._IMenuEvents_RClickEvent) Handles AxExMenu1.RClick  
    Debug.WriteLine(AxExMenu1.item(e.iD).Caption)  
End Sub
```

The following C# sample displays the caption of the item being right clicked:

```
private void axExMenu1_RClick(object sender, AxEXMENU.Lib._IMenuEvents_RClickEvent e)
```

```
{  
    System.Diagnostics.Debug.WriteLine(axExMenu1[e.iD].Caption);  
}
```

The following VFP sample displays the caption of the item being right clicked:

```
*** ActiveX Control Event ***  
LPARAMETERS id  
  
with thisform.ExMenu1  
    wait window nowait .Item(id).Caption  
endwith
```

event Select (ID as Long)

Occurs when an item is selected by clicking or by pressing RETURN key.

Type	Description
ID as Long	A long expression that indicates the item's identifier.

Use the Select event to notify your application that a new item was selected. Use the [Item](#) property to retrieve the item giving its identifier. Use the [SelectOn](#) property to specify whether the control selects an item if the user presses or releases the mouse button. Use the [Caption](#) property to specify the caption of the item. The control sends the WM_COMMAND message to the parent window, where the wParam is the ID (identifier) of the item being clicked.

Syntax for Select event, **/NET** version, on:

```
C# private void Select(object sender,int ID)
{
}
```

```
VB Private Sub Select(ByVal sender As System.Object,ByVal ID As Integer) Handles
Select
End Sub
```

Syntax for Select event, **/COM** version, on:

```
C# private void Select(object sender, AxEXMENUMLib._IMenuEvents_SelectEvent e)
{
}
```

```
C++ void OnSelect(long ID)
{
}
```

```
C++ Builder void __fastcall Select(TObject *Sender,long ID)
{
}
```

```
Delphi procedure Select(ASender: TObject; ID : Integer);
begin
```

```
end;
```

Delphi 8
(.NET
only)

```
procedure Select(sender: System.Object; e:  
AxEXMENULib._IMenuEvents_SelectEvent);  
begin  
end;
```

Powe...

```
begin event Select(long ID)  
end event Select
```

VB.NET

```
Private Sub Select(ByVal sender As System.Object, ByVal e As  
AxEXMENULib._IMenuEvents_SelectEvent) Handles Select  
End Sub
```

VB6

```
Private Sub Select(ByVal ID As Long)  
End Sub
```

VBA

```
Private Sub Select(ByVal ID As Long)  
End Sub
```

VFP

```
LPARAMETERS ID
```

Xbas...

```
PROCEDURE OnSelect(oExMenu,ID)  
RETURN
```

Syntax for Select event, **ICOM** version (others), on:

Java...

```
<SCRIPT EVENT="Select(ID)" LANGUAGE="JScript">  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function Select(ID)  
End Function  
</SCRIPT>
```

Visual
Data...

```
Procedure OnComSelect Integer IID  
Forward Send OnComSelect IID
```

```
End_Procedure
```

Visual
Objects

```
METHOD OCX_Select(ID) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_Select(int _ID)  
{  
}
```

XBasic

```
function Select as v (ID as N)  
end function
```

dBASE

```
function nativeObject_Select(ID)  
return
```

In VFP9, if using the `_SCREEN.AddObject` on modal forms, the `Select` may not be fired, instead you can use the `BINDEVENT` to collect the `WM_COMMAND` messages as shown in the following sample. The `IParam` of the `WM_COMMAND` message is 0 for the `Select` event.

```
*>>>> This is a replacement for the Select event using the BINDEVENT command  
LOCAL _aka  
_aka = CREATEOBJECT("AKA")  
BINDEVENT(_SCREEN.hWnd, 273, _aka, "myselect" )  
*<<<< BINDEVENT
```

where the `AKA` object could be as:

```
DEFINE class AKA as Custom  
  
function myselect(m,h,w,p)  
    MESSAGEBOX(STR(w),64, "Your command's identifier is:")  
    return 0  
endfunc  
  
ENDDEFINE
```

The following VB sample displays the caption of the item being selected:

```
Private Sub ExMenu1_Select(ByVal ID As Long)
    ' The user has selected an item
    Debug.Print "You have selected '" & ExMenu1(ID).Caption & "'"
End Sub
```

The following Javascript sample displays the caption of the item being selected:

```
<SCRIPT FOR="ExMenu1" EVENT="Select(id)" LANGUAGE="javascript">

obj = document.getElementById ( "ExMenu1" );
window.alert(obj.Item(id).Caption);

</SCRIPT>
```

The following C++ sample displays the caption of the item being selected:

```
void OnSelectExmenu1(long ID)
{
    OutputDebugString( m_menu.GetItem( COleVariant(ID) ).GetCaption() );
}
```

The following VB.NET sample displays the caption of the item being selected:

```
Private Sub AxExMenu1_SelectEvent(ByVal sender As System.Object, ByVal e As
AxEXMENULib._IMenuEvents_SelectEvent) Handles AxExMenu1.SelectEvent
    Debug.WriteLine(AxExMenu1.item(e.iD).Caption)
End Sub
```

The following C# sample displays the caption of the item being selected:

```
private void axExMenu1_SelectEvent(object sender,
AxEXMENULib._IMenuEvents_SelectEvent e)
{
    System.Diagnostics.Debug.WriteLine(axExMenu1[e.iD].Caption);
}
```

The following VFP sample displays the caption of the item being selected:

```
*** ActiveX Control Event ***
LPARAMETERS id
```

```
with thisform.ExMenu1.Item(id)
    wait window nowait .Caption
endwith
```