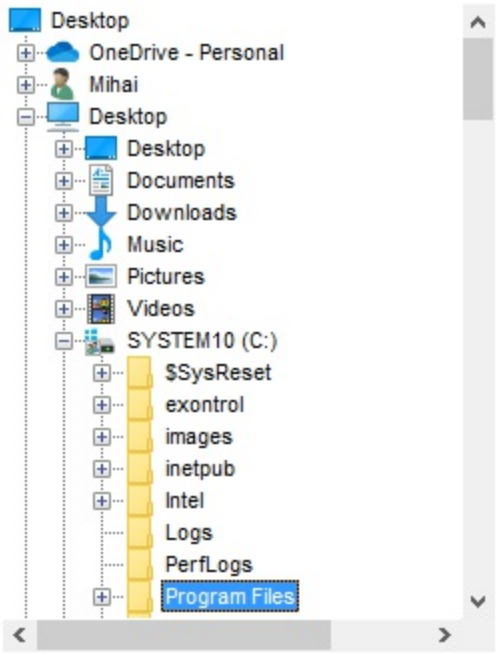
ExFolderView

Exontrol's new eXFolderView component provides a folder list view which is identical to the left side of your Windows Explorer. Using eXFolderView you can easily present a list of folders to your users. There are a number of properties which can be used to enable special features such as checkboxes, buttons, and lines between folders. eXFolderView can be used in conjunction with Exontrol's eXShellView to create applications which have complete - or limited - explorer capabilities. Also, the eXFolderView provides a complete list of events, giving you complete control over what happens as the user selects and clicks on folders. And eXFolderView provides a Folders collection containing Folder objects which provide useful information such as PathName, ShareName, and DisplayName.

note The eXShellView and eXFolderVew controls adds Windows-Explorer functionality (with the same look and behavior as your Explorer) to your forms. The main difference between eXFileView and eXShellView or eXFolderView, is that eXFileView can customize groups of files or folders with specified colors, fonts or icons, and the eXShellView and eXFolderVew uses the Windows system to create the views, and so the look and behavior is exactly like you would run your Windows Explorer in your form.



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 AppearanceEnum

Specifies the control's appearance. Use the Appearance property to specify the control's appearance.

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

constants AttributesEnum

The AttributesEnum type indicates attributes for a folder. The Attribute property indicates the folder's attribute.

Name	Value	Description
CanCopy	1	The specified file objects or folders can be copied
CanMove	2	The specified file objects or folders can be moved
CanLink	4	It is possible to create shortcuts for the specified file objects or folders
CanRename	16	The specified file objects or folders can be renamed
CanDelete	32	The specified file objects or folders can be deleted
HasPropSheet	64	The specified file objects or folders have property sheets
DropTarget	256	The specified file objects or folders are drop targets
Shortcut	65536	The specified file objects are shortcuts.
Share	131072	The specified folders are shared.
ReadOnly	262144	The specified file objects or folders are read-only
Hidden	524288	The specified file objects are hidden
HasSubfolder	-2147483648	The specified folders have subfolders
IsFileSysAncestor	268435456	The specified folders contain one or more file system folders
IsFolder	536870912	The specified items are folders
IsFileSystem	1073741824	The specified folders or file objects are part of the file system
Validate	16777216	Validate cached information
Removable	33554432	The specified file objects or folders are on removable media
IsCompressed	67108864	The specified items are compressed
IsBrowsable	134217728	The specified items can be browsed in place
NonEnumerated	1048576	The items are nonenumerated items
NewContent	2097152	The objects contain new content

constants AttributesMask

The AttributesMask type specifies different masks for attributes of the folder. The Attributes property retrieves groups of attributes based on the giving mask.

Name	Value	Description
AllAttributes	-1	All flags
CapabilityAttributes	375	This flag is a mask for the capability flags
DisplayAttributes	983040	This flag is a mask for the display attributes
ContentsAttributes	-2147483648	This flag is a mask for the contents attributes
MiscellaneousAttributes	-1048576	This flag is a mask for the Miscellaneous attributes

constants BordersEnum

Specifies the control's border. The BorderStyle property indicates the control's border.

Name	Value	Description
None	0	No border
FixedSingle	1	Single-line border

constants ScrollBarsEnum

The ScrollBars type indicates the type of scrollbars that the control may display when required. The Scrollbars property specifies whether the control should add horizontal or vertical scroll bars when they required.

Name	Value	Description
ScrollNone	0	No scroll bars are shown
Horizontal	1	Only horizontal scroll bars are shown
Vertical	2	Only vertical scroll bars are shown
Both	3	Both horizontal and vertical scroll bars are shown.

constants SpecialFolderPathEnum

Indicates special folders.

Name	Value	Description
Desktop	0	Windows Desktop virtual folder that is the root of the namespace.
Internet	1	Virtual folder representing the Internet
Programs	2	File system directory that contains the user's program groups (which are also file system directories).
ControlPanel	3	Virtual folder containing icons for the Control Panel applications
Printers	4	Virtual folder containing installed printers
Personal	5	File system directory that serves as a common repository for documents.
Favorites	6	File system directory that serves as a common repository for the user's favorite items.
Startup	7	File system directory that corresponds to the user's Startup program group. The system starts these programs whenever any user logs onto Windows NT or starts Windows 95.
Recent	8	File system directory that contains the user's most recently used documents.
SendTo	9	File system directory that contains Send To menu items.
Recycled	10	Virtual folder containing the objects in the user's Recycle Bin
StartMenu	11	File system directory containing Start menu items.
DesktopDir	16	File system directory used to physically store file objects on the desktop (not to be confused with the desktop folder itself). A common path is C:WINNTProfilesusernameDesktop
MyComputer	17	My Computer virtual folder containing everything on the local computer: storage devices, printers, and Control Panel. The folder may also contain mapped network drives
		Network Neighborhood virtual folder representing

Network	18	the top level of the network hierarchy
NetHood	19	File system directory containing objects that appear in the network neighborhood. A common path is C:WINNTProfilesusername etHood
Fonts	20	Virtual folder containing fonts.
Templates	21	File system directory that serves as a common repository for document templates
CommonStartMenu	22	File system directory that contains the programs and folders that appear on the Start menu for all users. A common path is C:WINNTProfilesll UsersStart Menu. Valid only for Windows NTŽ systems
CommonPrograms	23	File system directory that contains the directories for the common program groups that appear on the Start menu for all users. A common path is c:WINNTProfilesll UsersStart MenuPrograms. Valid only for Windows NTŽ systems
CommonStartup	24	File system directory that contains the programs that appear in the Startup folder for all users. A common path is C:WINNTProfilesll UsersStart MenuProgramsStartup. Valid only for Windows NTŽ systems
CommonDesktopDir	25	File system directory that contains files and folders that appear on the desktop for all users. A common path is C:WINNTProfilesll UsersDesktop. Valid only for Windows NTŽ systems
AppData	26	File system directory that serves as a common repository for application-specific data. A common path is C:WINNTProfilesusernamepplication Data
PrintHood	27	File system directory that serves as a common repository for printer links. A common path is C:WINNTProfilesusernamePrintHood
AltStartup	29	File system directory that corresponds to the user's nonlocalized Startup program group
CommonAltStartup	30	File system directory that corresponds to the nonlocalized Startup program group for all users. Valid only for Windows NTŽ systems
		File system directory that serves as a common

CommonFavorites	31	repository for all users' favorite items. Valid only for Windows NTŽ systems
InternetCache	32	File system directory that serves as a common repository for temporary Internet files. A common path is C:WINNTProfilesusername emporary Internet Files
Cookies	33	File system directory that serves as a common repository for Internet cookies. A common path is C:WINNTProfilesusernameCookies
History	34	File system directory that serves as a common repository for Internet history items

ExFolderCombo 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: {8AE82D12-B7B7-45E4-A795-76F7E7189F62}. The object's program identifier is: "Exontrol.FolderCombo". The /COM object module is: "ExFolderView.dll"

The ExFolderCombo control acts like a drop down combo box control, and it contains shell folders. By default the ExFolderCombo control contains the following folders: Desktop, My Computer, Drives and Network. The ExFolderCombo control is intended to be used with the ExFolderView or ExShellView controls. When it is used this way, the control displays the selected folder or the browsed folder in the caption. When dropped, the control displays a tree that contains the parent folders of the selected (browsed) one. To select a new folder in the ExFolderCombo control the user has to invoke the OpenedFolder. The property takes an ID, a path, or a reference to an Object that has a property called ID (for instance a Folder object of ExFolderView control, or a Folder object of ExShellView control).

Name	Description
Enabled	Retrieves or sets a value indicating whether the control can respond to user-generated events.
EventParam	Retrieves or sets a value that indicates the current's event parameter.
Font	Determines the Font object associated to the control
hwnd	Retrieves the window's handle.
OpenedFolder	Retrieves or sets an object that indicates the current opened folder.
SmallIcons	Retrieves or sets a value that indicates whether the control displays small icons or large icons.
Version	Retrieves the version of the control.

property ExFolderCombo.Enabled as Boolean

Retrieves or sets a value indicating whether the control can respond to user-generated events.

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

If set to True, the ExFolderCombo control will act normally. If this property is set to False, the ExFolderCombo control will not react to user input (mouse, keyboard, etc.).

property ExFolderCombo.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.

property ExFolderCombo.Font as IFontDisp

Determines the Font object associated to the control

Type	Description
IFontDisp	A Font object to be used in the combo box.

The Font property specifies the font to display folders in the combobox.

property ExFolderCombo.hwnd as Long

Retrieves the control's window handle.

Type	Description
Long	A long expression that indicates the control's window handle.

The Microsoft Windows operating environment identifies each form and control in an application by assigning it a handle, or hWnd. The hWnd property is used with Windows API calls. Many Windows operating environment functions require the hWnd of the active window as an argument.

property ExFolderCombo.OpenedFolder as Variant

Retrieves or sets an object that indicates the current opened folder.

Type	Description
Variant	A variant expression that specifies a folder (either a path name or a value from an ID property).

Retrieves or sets the new opened folder. The property accepts a path or an ID value. For instance, the following sample shows how to set the opened folder to C:\Temp:

```
ExFolderCombo1.OpenedFolder = "C:\Temp"
```

The following VB sample shows how to link a the drop down control with a EXFolderView control:

```
Private Sub ExFolderCombo1_NewFolderOpened()
   ExFolderView1.SelectedFolder = ExFolderCombo1.OpenedFolder
End Sub
```

The following C# sample shows how to link a the drop down control with a EXFolderView control:

```
private void axExFolderCombo1_NewFolderOpened(object sender, EventArgs e)
{
   axExFolderView1.SelectedFolder = axExFolderCombo1.OpenedFolder;
}
```

The following VB.NET sample shows how to link a the drop down control with a EXFolderView control:

```
Private Sub AxExFolderCombo1_NewFolderOpened(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles AxExFolderCombo1.NewFolderOpened
   AxExFolderView1.SelectedFolder = AxExFolderCombo1.OpenedFolder
End Sub
```

The following VC sample shows how to link a the drop down control with a EXFolderView control:

```
void OnNewFolderOpenedFoldercombo1()
{
```

```
    if ( IsWindow( m_folderCombo.m_hWnd ) && IsWindow( m_folderView.m_hWnd ) )
        m_folderView.SetSelectedFolder( m_folderCombo.GetOpenedFolder() );
}
```

The following VFP sample shows how to link a the drop down control with a EXFolderView control:

```
thisform.ExFolderView1.SelectedFolder = thisform.ExFolderCombo1.OpenedFolder
```

property ExFolderCombo.SmallIcons as Boolean

Retrieves or sets a value that indicates whether the control displays small icons or large icons.

Type	Description
Boolean	A Boolean expression that specifies whether the control displays small icons or large icons.

By default, the SmallIcons property is True, which indicates that the control display small-icons (16x16 icon-size). The SmallIcons property specifies whether the control displays small or large icons. The SmallIcons property On True, determines the control to display large-icons (32x32 icon-size).

property ExFolderCombo.Version as String

Retrieves the version of the control.

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

The version property specifies the control's version.

ExFolderView 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: {10670A99-FCCC-415C-8127-176332842618}. The object's program identifier is: "Exontrol.FolderView". The /COM object module is: "ExFolderView.dll"

Exontrol's new ExFolderView component provides a folder list view which is identical to the left side of your Windows Explorer. Using ExFolderView you can easily present a list of folders to your users. There are a number of properties which can be used to enable special features such as checkboxes, buttons, and lines between folders. ExFolderView can be used in conjunction with Exontrol's eXShellView to create applications which have complete - or limited - explorer capabilities. Also, the ExFolderView provides a complete list of events, giving you complete control over what happens as the user selects and clicks on folders. And ExFolderView provides a Folders collection containing Folder objects which provide useful information such as PathName, ShareName, and DisplayName. The control supports the following properties and methods:

Name	Description
AllowDropFiles	Indicates if the control accepts dropping files.
Appearance	Returns or sets a value that determines the appearance of the object.
AttachTemplate	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
AutoUpdate	Retrieves or sets a value indicating whether the control refreshes its content when a shell object was changed, moved, or renamed.
BackColor	Specifies the control's background color.
BorderStyle	Retrieves or sets the border style of the control.
CanRename	Retrieves or sets a value indicating whether the control add Rename context menu.
DisplayShareName	Indicating whether the control displays the share folder name.
DropFilesCount	Retrieves the count of dropped files.
DropFilesPathName	Retrieves the dropped files given an index,
Enabled	Returns or sets a value that determines whether a control can respond to user-generated events.
EnableShellMenu	Enables or disables the control's context menu.
EnsureVisible	Ensures a specified ExShellFolder object , path or special folder is visible.

EventParam	Retrieves or sets a value that indicates the current's event parameter.
ExecuteTemplate	Executes a template and returns the result.
ExploreFromHere	Retrieves or sets a path that indicates the root folder used to display the hierarchy.
FirstVisibleFolder	Retrieves or sets the first visible folder.
FolderFromPoint	Retrieves the ExShellFolder object from the point.
FoldersCheck	Gets a collection of folders being checked.
Font	Determines the Font object associated to the control
ForeColor	Specifies the control's foreground color.
HasButtons	Adds a (+/-) button to the left side of each parent item
HasCheckBoxes	Retrieves or sets a value indicating whether the folder has associated a checkbox.
HasLines	Enhances the graphic representation of a tree control's hierarchy by drawing lines that link child items to their corresponding parent item.
HasLinesAtRoot	Link items at the root of the hierarchy.
HiddenFolders	Retrieves or sets a value indicating whether the control displays the hidden folders.
HideSelection	Returns a value that determines whether selected item appears highlighted when a control loses the focus.
HorizontalHeight	Retrieves the width of the height scroll bar.
HorizontalOffset	Indicates the horizontal scroll position.
HorizontalOversize	Indicates the horizontal oversize value.
hwnd	Retrieves the window's handle.
IconsVisible	Determines if the control displays the icons.
IncludeAttributeMask	Retrieves or sets a value that determines the attribute mask used to enumerate the objects.
IncludeFolder	Retrieves or sets a value indicating whether the control fires IncludeFolder event before inserting folders to the tree.
ItemHeight	Gets or sets the height to display the folders/files of the control.
MouseIcon	This property identifies mouse icon when MousePointer is Custom.

MousePointer	Returns or sets a value indicating the type of mouse pointer displayed when the mouse is over a particular part of an object at run time.
OverlayIcons	Retrieves or sets a value indicating whether the control displays the overlay icons.
PartialCheck	Retrieves or sets a value indicating whether the control accepts partial-check feature.
Refresh	Refreshes the control.
Scrollbars	Returns value indicating whether an object has horizontal or vertical scroll bars
SelectedFolder	Retrieves or sets the selected folder.
ShellFolder	Constructs a ExShellFolder object given a path or a special folder constant.
SmallIcons	Retrieves or sets a value that indicates whether the control displays small icons or large icons.
SpecialFolderPath	Gets a path given a special folder constant.
Template	Specifies the control's template.
TemplateDef	Defines inside variables for the next Template/ExecuteTemplate call.
TemplatePut	Defines inside variables for the next Template/ExecuteTemplate call.
Version	Retrieves the version of the control.
VerticalOffset	Indicates the vertical scroll position.
VerticalOversize	Indicates the vertical oversize value.
VerticalWidth	Retrieves the width of the vertical scroll bar.
VisibleCount	Counts the visible folders.

property ExFolderView.AllowDropFiles as Boolean

Indicates if the control accepts dropping files.

Type	Description
Boolean	A boolean expression that specifies whether the control accepts dragged files/folders from other applications.

By default, the AllowDropFiles property is False. The AllowDropFiles property determines whether or not the control will accept files dragged-and-dropped from another application (such as Explorer). The DropFiles event notifies your application that the user just dragged some folders on your control. Use the DropFilesCount property to count the files being dropped. Use the DropFilesPathName property to retrieve the path of the dropping folder.

property ExFolderView.Appearance as AppearanceEnum

Returns or sets a value that determines the appearance of the object.

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

Use the Appearance property to change the control's appearance. Use the BorderStyle property to specify the control's borders.

method ExFolderView.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 ExFolderView1_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 ExFolderView.AutoUpdate as Boolean

Retrieves or sets a value indicating whether the control refreshes its content when a shell object was changed, moved, or renamed.

Type	Description
Boolean	A Boolean expression that specifies whether the control gets automatically updated once a folder is changed, renamed or moved.

the control receives notification messages each time the directory structure is changed, and so can update its list, if this property is True. If the structure is changed, the control updates itself accordingly. For example, if a user starts Explorer and renames a directory, and this property is set to True, change will take effect also in the control. If set to False, the control is not changed until the user manually forces refreshing the control. The control fires the FolderUpdate event when the control requires refreshing based on change notification on directory structure.

property ExFolderView.BackColor as Color

Specifies the control's background color.

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

The BackColor property specifies the control's background color. Use the ForeColor property to change the control's foreground color.

property ExFolderView.BorderStyle as BordersEnum

Retrieves or sets the border style of the control.

Type	Description
BordersEnum	A BordersEnum expression that indicates the control's border.

Use the BorderStyle property to specify the control's borders. Use the Appearance property to change the control's appearance.

property ExFolderView.CanRename as Boolean

Retrieves or sets a value indicating whether the control add Rename context menu.

Type	Description
Boolean	A Boolen expression that specifies whether the control allows renaming the folders.

By default, the CanRename property is False. If the CanRename property is True, for folders or files that support renaming, the control inserts into its context menu a new menu item *Rename*. If the user selects it, the control will display an edit control where he will be able to change the name of the folder or file. Use the EnableShellMenu property to disable control's context menu, when the user right clicks a folder. Use the QueryContextMenu event to add new items to the folder's context menu. Use the InvokeRename method to rename a folder at run-time.

property ExFolderView.DisplayShareName as Boolean

Indicating whether the control displays the share folder name.

Type	Description
Boolean	A Boolen expression that specifies whether the control displays the folder's share name.

Setting this property to True will cause the control to display share-names for all folders that are shared. If property is set to False, shared names are not shown, but folder is displayed anyway with it's real name. For example, let's suppose that we a have folder "C:\Documents and Settings\Administrator\My Documents" and it is shared with name "docs". If DisplayShareName property is set to True, the control will display text "My Documents [docs]" when this folder is listed.

property ExFolderView.DropFilesCount as Long

Retrieves the count of dropped files.

Type	Description
Long	A long expression that counts the files being droped.

Use the DropFilesCount property to count the files being dropped. Use the DropFilesPathName property to retrieve the path of the dropping folder. The control fires the DropFiles event to notify your application that some files were dropped. The AllowDropFiles property determines whether or not the control will accept files dragged-and-dropped from another application (such as Explorer).

Here is a VB sample that lists the files dragged in the Immediate debugger window.

```
Private Sub ExFolderView1_DropFiles(ByVal ExShellFolder As
EXFOLDERVIEWLibCtl.IExShellFolder, ByVal Effect As Long)
  Dim I As Long
  For I = 0 To FolderView1.DropFilesCount - 1
    Debug.Print FolderView1.DropFilesPathName(I)
  Next I
  If (Effect & 1) = 1 Then
    Debug.Print "Copied to " & Folder.PathName
  Else
    Debug.Print "Moved to " & Folder.PathName
  End If
End Sub
```

property ExFolderView.DropFilesPathName (Index as Long) as String

Retrieves the dropped files given an index,

Type	Description
Index as Long	A Long expression that indicates the index of the folder being requested
String	A String expression that indicates the path being requested

Use the DropFilesPathName property to retrieve the path of the dropping folder. Use the DropFilesCount property to count the files being dropped. The control fires the DropFiles event to notify your application that some files were dropped. The AllowDropFiles property determines whether or not the control will accept files dragged-and-dropped from another application (such as Explorer).

Here is a VB sample that lists the files dragged in the Immediate debugger window.

```
Private Sub ExFolderView1_DropFiles(ByVal ExShellFolder As
EXFOLDERVIEWLibCtl.IExShellFolder, ByVal Effect As Long)
  Dim I As Long
  For I = 0 To FolderView1.DropFilesCount - 1
    Debug.Print FolderView1.DropFilesPathName(I)
  Next I
  If (Effect & 1) = 1 Then
    Debug.Print "Copied to " & Folder.PathName
  Else
    Debug.Print "Moved to " & Folder.PathName
  End If
End Sub
```

property ExFolderView.Enabled as Boolean

Returns or sets a value that determines whether a control can respond to user-generated events.

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

By default, the Enabled property is True. The Enabled property enables or disables the control.

property ExFolderView.EnableShellMenu as Boolean

Enables or disables the control's context menu.

Type	Description
Boolean	A Boolean expression that indicates whether the control provides a drop down context menu.

The control provides a drop down context menu, if the EnableShellMenu property is True. Right clicking on a folder results in showing the folder's context menu. If some item from this menu is selected, the control fires a pair of events to notify your application about this event. Before the menu item is executed, BeforeShellMenuCommand event is fired. After the menu item is executed, the AfterShellMenuCommand event is fired. Please note that Windows OS is a multitasking environment. That means that execution of menu item does not mean that this event is fired after the executed program (or command) has finished. Rather, immediately after execution of the menu item this event is fired. Use the QueryContextMenu event to add new items to the folder's context menu. The CanRename property retrieves or sets a value indicating whether the control add Rename context menu.

method ExFolderView.EnsureVisible (Folder as Variant)

Ensures a specified ExShellFolder object , path or special folder is visible.

Type	Description
Folder as Variant	A String expression that indicates a path, an ExShellFolder object, a long expression that indicates a special folder, being ensured that it fits the control's client area.

Use this method to make a folder visible. The Folder parameter can be any path, special folder, ExShellFolder or IShellFolder. Calling this method will, if necessary, open all parent folders until referred folder is shown.

property ExFolderView.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 ExFolderView.ExecuteTemplate (Template as String)

Executes a template and returns the result.

Type	Description
Template as String	A Template string being executed

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

Use the ExecuteTemplate property to returns the result of executing a template file. Use the Template property to execute a template without returning any result. Use the ExecuteTemplate property to execute code by passing instructions as a string (template string). For instance, you can use the EXPRINT.PrintExt = CONTROL.ExecuteTemplate("me") to print the control's content.

For instance, the following sample retrieves the the handle of the first visible item:

```
Debug.Print ExFolderView1.ExecuteTemplate("Items.FirstVisibleItem()")
```

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

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

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

The Template script is composed by lines of instructions. Instructions are separated by

"\n\r" (newline) characters.

An instruction can be one of the following:

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

The Template supports the following general functions:

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

property ExFolderView.ExploreFromHere as String

Retrieves or sets a path that indicates the root folder used to display the hierarchy.

Type	Description
String	A string expression that specifies the root folder for this control. If the ExploreFromHere property ends with "|reset" the control does not restore previously selected / expanded items. For instance, ExploreFromHere = "|reset" loads again the desktop, without selecting or expanding the previously selected / expanded items. The "reset" options must be after a "|" (pipe-character)

This property determines the root folder for the control. By default, if left empty, exploration starts at the desktop. Otherwise, the root folder is the path specified here.

property ExFolderView.FirstVisibleFolder as ExShellFolder

Retrieves or sets the first visible folder.

Type	Description
ExShellFolder	A Folder object that indicates the first visible folder in the control's list.

Property returns a Folder object that is currently positioned as upper-most visible folder. This may not be 'Desktop', or any root folder you selected. The value of this property depends upon what is currently shown inside the control. If the user moves scroll bars this value changes. Setting this value to some other folder will cause the control to open all parent folders of the folder, and position the folder to be the upper-most visible folder. However, there might be also folders above the referenced folder, they will become non-visible. However, moving scroll bars could make them visible. Use the ShellFolder property to create an ExShellFolder object based on the path.

As example, this code would set "C:\WINNT" as the most upper visible folder:

```
Private Sub Form_Load()
   ExFolderView1.FirstVisibleFolder = FolderView1.ShellFolder("C:\WINNT")
End Sub
```

property ExFolderView.FolderFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS) as ExShellFolder

Retrieves the ExShellFolder object from the point.

Type	Description
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in client coordinates.
ExShellFolder	A Folder from the cursor.

Use the FolderFromPoint property to get the item from the point specified by the {X,Y}. The X and Y coordinates are expressed in client coordinates, so a conversion must be done in case your coordinates are relative to the screen or to other window. **If the X parameter is -1 and Y parameter is -1 the ItemFromPoint property determines the handle of the folder from the cursor**. This property is used to determine what folder is displayed at given coordinates. If given coordinates are pointing to background and not particular folder, 'Nothing' is returned. Note that Visual Basic all coordinates represents in 'twips', and the control accepts coordinates in 'pixels'. Therefore, all twips coordinates should be scaled to pixels when calling this method. ScaleX and ScaleY methods will do the job for you.

Below example displays the folder from the cursor:

```
Private Sub ExFolderView1_MouseMove(ByVal Button As Integer, ByVal Shift As Integer,
ByVal X As Single, ByVal Y As Single)
  With ExFolderView1
    Dim f As EXFOLDERVIEWLibCtl.ExShellFolder
    Set f = .FolderFromPoint(-1, -1)
    If Not (f Is Nothing) Then
      Debug.Print f.DisplayName
    End If
  End With
End Sub
```

property ExFolderView.FoldersCheck as ExShellFolders

Gets a collection of folders being checked.

Type	Description
ExShellFolders	An ExShellFolders object that indicates a collection of checked folders.

If HasCheckBoxes property is set to True, each displayed folder has it's own check box. The AfterCheck event notifies your application whether a folder is checked or unchecked. Use the PartialCheck property to enable partial check feature. This property is used to find the folders that are currently checked.

property ExFolderView.Font as IFontDisp

Determines the Font object associated to the control

Type	Description
IFontDisp	A Font object being used to display the control's hierarchy.

Use the Font property to change the control's font. The BackColor property specifies the control's background color. Use the ForeColor property to change the control's foreground color.

property ExFolderView.ForeColor as Color

Specifies the control's foreground color.

Type	Description
Color	A Color expression that specifies the control's foreground color.

Use the ForeColor property to change the control's foreground color. The BackColor property specifies the control's background color. Use the Font property to specify the control's font.

property ExFolderView.HasButtons as Boolean

Adds a (+/-) button to the left side of each parent item

Type	Description
Boolean	A boolean expression that determine if (+/-) buttons are displayed for each folder.

If this property is set to True, for each folder that has subfolders a boxed '+' sign is showed, representing a button. Clicking on that button, folder gets expanded. Even if this property is set to False, and buttons are not shown, the user is still able to expand folders, using the keyboard, or by double clicking the folder.

property ExFolderView.HasCheckBoxes as Boolean

Retrieves or sets a value indicating whether the folder has associated a checkbox.

Type	Description
Boolean	A boolen expression that specifies whether the control displays a check box for each folder.

Setting the HasCheckBoxes property to True causes the control to show a check-box for each displayed folder. User can check or uncheck those check-boxes at run time. The AfterCheck event notifies your application whether a folder is checked or unchecked. Use the PartialCheck property to enable partial check feature. This way, if a folder is checked, it's parent folder is partial checked, if it contains un-checked folders, or full checked, if all child folders are checked. Use the FoldersCheck property to get the collection of checked folders, at run-time.

property ExFolderView.HasLines as Boolean

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

Type	Description
Boolean	A boolean expression that determine if lines are shown in control.

Setting this property to True, line will be drawn that connected related folders (for example, parent and child folders). It is used to graphically show directory structure. Setting this property to False, lines are not shown.

property ExFolderView.HasLinesAtRoot as Boolean

Link items at the root of the hierarchy.

Type	Description
Boolean	A boolean expression that determine if line is drawn between root foders and left edge of the control.

If this property is set to True, line is drawn between each root folder, and left edge of FolderView/X. Please note that the 'root' folder doesn't necessary need to be '\' folder. It might be 'My Computer', 'Destkop', etc. depending on the FirstVisibleFolder property

property ExFolderView.HiddenFolders as Boolean

Retrieves or sets a value indicating whether the control displays the hidden folders.

Type	Description
Boolean	A boolean expression that determine if folders marked 'hidden' are shown.

Use this property to specify if folders that have attribute 'hidden' are shown or not. Usually, Windows operating system sets on some special folder this attribute (Recycle Bin is a good example). If this property is set to False, such folders are not shown.

property ExFolderView.HideSelection as Boolean

Returns a value that determines whether selected item appears highlighted when a control loses the focus.

Type	Description
Boolean	A Boolean expression that indicates whether the control shows or hides the selection while the control loses the focus.

Use the HideSelection property to specify whether the control should display the selected folder in the control, even if it loses the focus.

property ExFolderView.HorizontalHeight as Long

Retrieves the width of the height scroll bar.

Type	Description
Long	A long expression that indicates the height of the control's horizontal scroll bar, if present.

This property returns the horizontal scroll bar's height. The HorisontalOffset property specifies the horizontal scroll position. The HorisontalOversize property specifies the maximum value for the horizontal scroll's position. Use the Scrollbars property to specify whether the control should add horizontal or vertical scroll bars when they required.

property ExFolderView.HorizontalOffset as Long

Indicates the horizontal scroll position.

Type	Description
Long	A long expression that specifies the position of the horizontal scroll bar.

The HorisontalOffset property specifies the horizontal scroll position. The HorizontalHeight property returns the horizontal scroll bar's height. The HorisontalOversize property specifies the maximum value for the horizontal scroll's position. Use the Scrollbars property to specify whether the control should add horizontal or vertical scroll bars when they required.

property ExFolderView.HorizontalOversize as Long

Indicates the horizontal oversize value.

Type	Description
Long	A long expression that specifies the maximum position for the control's horizontal scroll bar

The HorisontalOversize property specifies the maximum value for the horizontal scroll's position. The HorizontalOffset property specifies the horizontal scroll position. The HorizontalHeight property returns the horizontal scroll bar's height. Use the Scrollbars property to specify whether the control should add horizontal or vertical scroll bars when they required.

property ExFolderView.hwnd as Long

Retrieves the window's 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.

property ExFolderView.IconsVisible as Boolean

Determines if the control displays the icons.

Type	Description
Boolean	A Boolean expression that specifies whether the control displays the associated icons for each folder.

By default, the IconsVisible property is True, and so associated icons are displayed for each folder. Setting this property to 'False' will cause the control not to show icons left to folder names. Setting the OverlayIcons property to True causes the control to display Overlay-ed icons. The SmallIcons property specifies whether the control displays small or large icons.

property ExFolderView.IncludeAttributeMask as Long

Retrieves or sets a value that determines the attribute mask used to enumerate the objects.

Type	Description
Long	A long expression that specifies a mask to use

Setting the IncludeAttributeMask property determines the type of Folders that control displays. This can be quite selective. The value should be a combination of Folder item attribute values. A complete list of these values is found in the AttributesEnum type. Use the IncludeFolder property to customize the list of folders being displayed in the control.

Example, selecting folders with new items:

```
ExFolderView1.IncludeAttributeMask = NewContent Or IsFolder Or IsFileSystem
```

property ExFolderView.IncludeFolder as Boolean

Retrieves or sets a value indicating whether the control fires IncludeFolder event before inserting folders to the tree.

Type	Description
Boolean	A Boolean expression that specifies whether the control fires the IncludeFolder event to customize the list of folders being displayed.

By default, the IncludeFolder property is False. The IncludeFolder event notifies your application that a new folder is included in the control's list. Setting the IncludeAttributeMask property determines the type of Folders that control displays

property ExFolderView.ItemHeight as Long

Gets or sets the height to display the folders/files of the control.

Type	Description
Long	A long expression that specifies the height of the item.

By default, the ItemHeiht property is 20. Use the ItemHeight property to change the the height to display the folders/files of the control.

property ExFolderView.MouseIcon as IPictureDisp

This property identifies mouse icon when MousePointer is Custom.

Type	Description
IPictureDisp	A Picture object that indicates the cursor being displayed when the MousePointer property is custom.

Use the MouseIcon property to specify a custom cursor when the cursor hovers the control. Use the MousePointer property to specify a predefined cursor to be displayed while cursor hovers the control.

property ExFolderView.MousePointer as Long

Returns or sets a value indicating the type of mouse pointer displayed when the mouse is over a particular part of an object at run time.

Type	Description
Long	A long expression that indicates a predefined cursor.

Use the MousePointer property to specify a predefined cursor to be displayed while cursor hovers the control. Use the MouseIcon property to specify a custom cursor when the cursor hovers the control.

Following are the constants for mouse pointers:

Value	Description
0	(Default) Shape determined by the object
1	Arrow
2	Cross (cross-hair pointer)
3	I Beam
4	Icon (small square within a square)
5	Size (four-pointed arrow pointing north, south, east, and west)
6	Size NE SW (double arrow pointing northeast and southwest)
7	Size N S (double arrow pointing north and south)
8	Size NW, SE
9	Size E W (double arrow pointing east and west)
10	Up Arrow
11	Hourglass (wait)
12	No Drop
13	Arrow and hourglass
14	Arrow and question mark
15	Size all
16	Hand
99	Custom icon specified by the **MouseIcon** property.

property ExFolderView.OverlayIcons as Boolean

Retrieves or sets a value indicating whether the control displays the overlay icons.

Type	Description
Boolean	A boolean expression that specifies if Overlay-ed icons are shown (True), or not (False).

Overlay-ed icons are ones that are drawn over an existing icon. Windows uses Overlay-ed icons to notify the user that some item has special function or attribute. For example, shortcut icons have a small arrow in lower-left corner, shared folders have a hand that shows that folder is shared, etc. Setting the OverlayIcons property to True causes the control to display Overlay-ed icons. If this property is set to False, only original icons will be shown, without any Overlay-ed on it. Use the IconsVisible property to hide the associated icons for each folder. The SmallIcons property specifies whether the control displays small or large icons.

property ExFolderView.PartialCheck as Boolean

Retrieves or sets a value indicating whether the control accepts partial-check feature.

Type	Description
Boolean	A boolean expression that specifies whether the control partial check feature, and so, a check box can have three states: partial, checked or unchecked.

Use the PartialCheck property to enable partial check feature. This way, if a folder is checked, it's parent folder is partial checked, if it contains un-checked folders, or full checked, if all child folders are checked. Use the FoldersCheck property to get the collection of checked folders, at run-time. Setting the HasCheckBoxes property to True causes the control to show a check-box for each displayed folder. User can check or uncheck those check-boxes at run time. The AfterCheck event notifies your application whether a folder is checked or unchecked.

method ExFolderView.Refresh ()

Refreshes the control.

Type	Description

Calling Refresh method causes the control to repaint itself immediately. This method is useful when, for example, AutoUpdate property is set to False, and the current folder structure may have been changed. By default, the Refresh method is called when the user presses the F5 key.

property ExFolderView.Scrollbars as ScrollBarsEnum

Returns value indicating whether an object has horizontal or vertical scroll bars

Type	Description
ScrollBarsEnum	A ScrollBarsEnum expression that specifies whether the control adds horizontal and vertical scroll bars when they required.

Use the Scrollbars property to specify whether the control should add horizontal or vertical scroll bars when they required. The HorisontalOversize property specifies the maximum value for the horizontal scroll's position. The HorizontalOffset property specifies the horizontal scroll position. The HorizontalHeight property returns the horizontal scroll bar's height. The VerticalOversize property specifies the maximum value for the vertical scroll's position. The VerticalWidth property returns the vertical scroll bar's width. The VerticalOffset property specifies the vertical scroll position.

property ExFolderView.SelectedFolder as Variant

Retrieves or sets the selected folder.

Type	Description
Variant	A Folder object that indicates the control's selected folder.

Use the SelectedFolder property to specify a new selected folder. Use the ShellFolder or SpecialFolderPath property to build an ExShellFolder object.

The following VB sample shows how to link a the drop down control with a EXFolderView control:

```
Private Sub ExFolderCombo1_NewFolderOpened()
   ExFolderView1.SelectedFolder = ExFolderCombo1.OpenedFolder
End Sub
```

The following C# sample shows how to link a the drop down control with a EXFolderView control:

```
private void axExFolderCombo1_NewFolderOpened(object sender, EventArgs e)
{
   axExFolderView1.SelectedFolder = axExFolderCombo1.OpenedFolder;
}
```

The following VB.NET sample shows how to link a the drop down control with a EXFolderView control:

```
Private Sub AxExFolderCombo1_NewFolderOpened(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles AxExFolderCombo1.NewFolderOpened
   AxExFolderView1.SelectedFolder = AxExFolderCombo1.OpenedFolder
End Sub
```

The following VC sample shows how to link a the drop down control with a EXFolderView control:

```
void OnNewFolderOpenedFoldercombo1()
{
   if ( IsWindow( m_folderCombo.m_hWnd ) && IsWindow( m_folderView.m_hWnd ) )
      m_folderView.SetSelectedFolder( m_folderCombo.GetOpenedFolder() );
}
```

The following VFP sample shows how to link a the drop down control with a EXFolderView control:

```
thisform.ExFolderView1.SelectedFolder = thisform.ExFolderCombo1.OpenedFolder
```

property ExFolderView.ShellFolder (Path as Variant) as ExShellFolder

Constructs a ExShellFolder object given a path or a special folder contant.

Type	Description
Path as Variant	A String expression that indicates the folder's path
ExShellFolder	An ExShellFolder object being built based on the folder's path.

Supplying a folder's path name, such as "c:\windows" as the index argument Path will return the corresponding Folder object.

property ExFolderView.SmallIcons as Boolean

Retrieves or sets a value that indicates whether the control displays small icons or large icons.

Type	Description
Boolean	A Boolean expression that specifies whether the control displays small icons or large icons.

By default, the SmallIcons property is True, which indicates that the control display small-icons (16x16 icon-size). The SmallIcons property specifies whether the control displays small or large icons. The SmallIcons property On True, determines the control to display large-icons (32x32 icon-size). The IconsVisible property shows or hides the icons within the control. The OverlayIcons property shows or hides Overlay-ed icons.

property ExFolderView.SpecialFolderPath (specialFolder as SpecialFolderPathEnum) as String

Gets a path given a special folder constant.

Type	Description
specialFolder as SpecialFolderPathEnum	Constant value that specifies folder whose path is to be retrieved.
String	A String expression that retrieves the special folder path.

Windows has some folders that are marked as shell, or special. "Recycle bin" is one of them, "My Computer" also, etc. This property will return complete path to any of such folders, depending on value below. Please note that these folders don't necessarily need to be placed on the same place on every computer. Depending on installation, the drive Windows is located on, or user changes, these folders can be stored anywhere on disk. Because of this it's is advised to use this property to retrieve path to such folders, just to be sure that you are referring to same folder, no matter where your program will be used. Use the ShellFolder property to convert a path to an EnShellFolder object.

property ExFolderView.Template as String

Specifies the control's template.

Type	Description
String	A string expression that defines the control's template

The control's template uses the X-Script language to initialize the control's content. Use the Template property page of the control to update the control's Template property. Use the Template property to execute code by passing instructions as a string (template string). Use the ToTemplate property to generate the control's content to template format. Use the ExecuteTemplate property to get the result of executing a template script.

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

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

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

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

An instruction can be one of the following:

- **Dim** list of variables *Declares the variables. Multiple variables are separated by commas. (Sample: Dim h, h1, h2)*
- variable **=** property(list of arguments) *Assigns the result of the property to a variable. The "variable" is the name of a declared variable. The "property" is the property name*

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

The Template supports the following general functions:

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

The following sample loads the Orders table:

```
Dim rs
ColumnAutoResize = False
rs = CreateObject("ADOR.Recordset")
{
Open("Orders","Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program
Files\Exontrol\ExExFolderView\Sample\SAMPLE.MDB", 3, 3 )
}
DataSource = rs
```

property ExFolderView.TemplateDef as Variant

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
Variant	A string expression that indicates the Dim declaration, or any Object expression to be assigned to previously declared variables.

The TemplateDef property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters. It is known that programming languages such as **dBASE Plus or XBasic from AlphaFive**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters)* = *Value*, so our controls provide an alternative using the TemplateDef method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be Template or ExecuteTemplate property which can use the variable a and b being defined previously.

So, calling the TemplateDef property should be as follows:

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

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

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

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

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

```
   .Items.AddItem 2
End With
```

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

```
local Control,var_Column

Control = form.Activex1.nativeObject
// Control.Columns.Add("Column 1").Def(4) = 255
var_Column = Control.Columns.Add("Column 1")
with (Control)
   TemplateDef = [Dim var_Column]
   TemplateDef = var_Column
   Template = [var_Column.Def(4) = 255]
endwith
Control.Columns.Add("Column 2")
Control.Items.AddItem(0)
Control.Items.AddItem(1)
Control.Items.AddItem(2)
```

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

```
Dim Control as P
Dim var_Column as P

Control = topparent:CONTROL_ACTIVEX1.activex
' Control.Columns.Add("Column 1").Def(4) = 255
var_Column = Control.Columns.Add("Column 1")
Control.TemplateDef = "Dim var_Column"
Control.TemplateDef = var_Column
Control.Template = "var_Column.Def(4) = 255"

Control.Columns.Add("Column 2")
Control.Items.AddItem(0)
Control.Items.AddItem(1)
Control.Items.AddItem(2)
```

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

The TemplateDef, Template and ExecuteTemplate support x-script language (Template script of the Exontrols), like explained bellow:

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

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

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

The x-script may uses constant expressions as follow:

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

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

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

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

method ExFolderView.TemplatePut (NewVal as Variant)

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
NewVal as Variant	A string expression that indicates the Dim declaration, or any Object expression to be assigned to previously declared variables.

The TemplatePut method / TemplateDef property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters. It is known that programming languages such as **dBASE Plus or XBasic from AlphaFive**, does not support setting a property with multiple parameters. In other words, these programming languages does not support something like *Property(Parameters) = Value*, so our controls provide an alternative using the TemplateDef / TemplatePut method. The first call of the TemplateDef should be a declaration such as "Dim a,b" which means the next 2 calls of the TemplateDef defines the variables a and b. The next call should be Template or ExecuteTemplate property which can use the variable a and b being defined previously.

The TemplateDef, TemplatePut, Template and ExecuteTemplate support x-script language (Template script of the Exontrols), like explained bellow:

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

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

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

Columns.Add("Column1").HeaderBackColor = RGB(255,0,0), adds a new column and changes the column's header back color.

The x-script may uses constant expressions as follow:

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

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

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

property ExFolderView.Version as String

Retrieves the version of the control.

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

The version property specifies the control's version.

property ExFolderView.VerticalOffset as Long

Indicates the vertical scroll position.

Type	Description
Long	A long expression that specifies the position of the vertical scroll bar.

The VerticalOffset property specifies the vertical scroll position. The VerticalHeight property returns the vertical scroll bar's width. The VerticalOversize property specifies the maximum value for the vertical scroll's position. Use the Scrollbars property to specify whether the control should add vertical or vertical scroll bars when they required.

property ExFolderView.VerticalOversize as Long

Indicates the vertical oversize value.

Type	Description
Long	A long expression that indicates the maximum position for the control's vertical scroll bar.

The VerticalOversize property specifies the maximum value for the vertical scroll's position. The VerticalWidth property returns the vertical scroll bar's width. The VerticalOffset property specifies the vertical scroll position. Use the Scrollbars property to specify whether the control should add vertical or vertical scroll bars when they required.

property ExFolderView.VerticalWidth as Long

Retrieves the width of the vertical scroll bar.

Type	Description
Long	A long expression that indicates the width of the control's vertical scroll bar, in pixels.

The VerticalWidth property returns the vertical scroll bar's width. The VerticalOffset property specifies the vertical scroll position. The VerticalOversize property specifies the maximum value for the vertical scroll's position. Use the Scrollbars property to specify whether the control should add vertical or vertical scroll bars when they required.

property ExFolderView.VisibleCount as Long

Counts the visible folders.

Type	Description
Long	A long expression that specifies the number of folders being displayed in the control's client area.

This property will return the exact number of currently folders which are currently visible within the control's borders

ExShellFolder object

The ExShellFolder object supports the following properties and methods:

Name	Description
Attribute	Asks for a specific attribute.
Attributes	Retrieves the attributes of the source.
Check	Checks if the folder checked or unchecked.
DisplayName	Retrieves the name displayed in the tree for the source.
Expanded	Specifies whether the folder is expanded or collapsed.
FolderPath	Retrieves the folder path, using the API SHGetFolderPath.
Folders	Retrieves a ExShellFolder object collection representing the subfolders of source.
Handle	Retrieves the handle of the source.
ID	Retrieves the ExShellFolder's ITEMIDLIST as an safe array.
InvokeCommand	Invokes a specified command from the object's context menu.
InvokeRename	Performs the rename operation
Loaded	Retrieves a value that indicates whether the control has loaded the child folders of the source.
Name	Retrieves the name of the source.
Parent	Retrieves the parent folder of the source.
PartialCheck	Checks if the source is partial-checked or not.
PathName	Retrieves the path of the source.
ShareName	Retrieves the share folder name.
UserData	Associates an extra data.

property ExShellFolder.Attribute (Attribute as AttributesEnum) as Boolean

Asks for a specific attribute.

Type	Description
Attribute as AttributesEnum	An Attribute value being requested
Boolean	A Boolean expression that indicates whether the folder has the specified attribute

This property specifies the custom flags for a folder object.Not all visible objects inside FolderView/X are the same. The differences are shown in the table below. Use this property to determine if the current Folder Object is a short-cut, is visible, etc.

property ExShellFolder.Attributes (Mask as AttributesMask) as Long

Retrieves the attributes of the source.

Type	Description
Mask as AttributesMask	A long expression that indicates the mask of the attributes being requested.
Long	A long expression that indicates the group of attributes being requested.

This property returns one or more of an object's attributes. Depending on the attribute types we're interested in, use a value from the AttributesMask table. The return value will be a combination of all the object's attributes.

property ExShellFolder.Check as Boolean

Checks if the folder checked or unchecked.

Type	Description
Boolean	A boolean expression that indicates whether the folder is check or unchecked.

The Check property determines whether the folder is checked or unchecked. Use the PartialCheck property to determine whether the folder is partially checked. Use the HasCheckBoxes property to assign a check box to each folder in the control's list. Use the PartialCheck property to enable partial check feature. This way, if a folder is checked, it's parent folder is partial checked, if it contains un-checked folders, or full checked, if all child folders are checked.

property ExShellFolder.DisplayName as String

Retrieves the name displayed in the tree for the source.

Type	Description
String	A String expression that indicates the name of the folder being displayed in the control's list.

Retrieves or sets the display name of the current folder. This is the same as when the control uses in drawing the associated item.

property ExShellFolder.Expanded as Boolean

Specifies whether the folder is expanded or collapsed.

Type	Description
Boolean	A Boolean expression that indicates whether the folder is expanded or collapsed.

Use the Expanded property to specify whether a folder is expanded or collapsed.

property ExShellFolder.FolderPath as String

Retrieves the folder path, using the API SHGetFolderPath.

Type	Description
String	A string expression that specifies the path of the folder.

This property returns the path associated with this folder. To do so, the component uses the ID property to retrieve the path from the file system.

property ExShellFolder.Folders as Variant

Retrieves a ExShellFolder object collection representing the subfolders of source.

Type	Description
Variant	An ExShellFolders object that indicates the collection of sub-folders.

Folders mirror the structure of folders as seen in the Windows Explorer. Some folders may contain other folders. The ExShellFolders provides access to contained folders. If a given folder has no subfolder then SomeFolder.Folders.Count will be zero.

property ExShellFolder.Handle as LONG_PTR

Retrieves the handle of the source.

Type	Description
LONG_PTR	A Handle expression.

Retrieves the handle of the Folder which uniquely identifies it in the control's list. This is different from the ID property.

property ExShellFolder.ID as Variant

Retrieves the ExShellFolder's ITEMIDLIST as an safe array.

Type	Description
Variant	A Variant expression that indicates the identifier of the folder.

Windows owns the ITEMIDLIST structure. Use this property when you want to link ExShellView or ExFolderCombo controls with ExFolderView control. For instance the following sample shows how to link a ExFolderCombo control with a ExFolderView control:

```
Private Sub FolderComboX1_NewFolderOpened()
  ExFolderView1.SelectedFolder = ExFolderComboX1.OpenedFolder
End Sub

Private Sub ExFolderCombo1_NewFolderOpened()
  ExFolderComboX1.OpenedFolder = NewFolder.ID
End Sub
```

This property is read-only and only available at run-time

method ExShellFolder.InvokeCommand (CommandName as String)

Invokes a specified command from the object's context menu.

Type	Description
CommandName as String	A String expression that indicates the name of the command being executed.

The InvokeCommand method executes a command from the folder's context menu. Use the InvokeRename method to rename a folder at runtime. Use the SelectedFolder property to get the selected folder in the control.

The following VB sample displays the object's Properties dialog, when the user presses the F2 key:

```
Private Sub ExFolderView1_KeyDown(KeyCode As Integer, Shift As Integer)
   If (KeyCode = vbKeyF2) Then
      ExFolderView1.SelectedFolder.InvokeCommand ("Properties")
   End If
End Sub
```

method ExShellFolder.InvokeRename ()

Performs the rename operation

Type	Description

Call the InvokeRename method to call renaming an folder or a file, at run-time. The CanRename property should be set on True, before performing the InvokeRename method, else it has no effect. Use the SelectedFolder property to get the selected folder in the control. The rename operation starts only if the selected shell object supports renaming. For instance, if you try to rename the My Computer folder, it is not allowed, since it doesn't support renaming. The control is focused, when InvokeRename method is called. Use the InvokeCommand method to execute a command from the object's context menu.

The following VB sample starts renaming the selected folder, when the user presses the F2 key:

```
Private Sub ExFolderView1_KeyDown(KeyCode As Integer, Shift As Integer)
   If (KeyCode = vbKeyF2) Then
      ExFolderView1.SelectedFolder.InvokeRename
   End If
End Sub
```

property ExShellFolder.Loaded as Boolean

Retrieves a value that indicates whether the control has loaded the child folders of the source.

Type	Description
Boolean	A boolean expression that indicates whether the control has loaded the child folders of the source.

The Loaded property returns true, if the child folders of the source are loaded.

property ExShellFolder.Name as String

Retrieves the name of the source.

Type	Description
String	A String expression that indicates the name of the folder.

This is the name of the Folder. The DisplayName and Name could be different. An example is when you have a Folder which stores link to the C:. The DisplayName is the label of the disk plus (C:), and the Name of it is C:

property ExShellFolder.Parent as ExShellFolder

Retrieves the parent folder of the source.

Type	Description
ExShellFolder	An ExShellFolder object that specifies the parent folder of the current folder.

The Parent property retrieves the parent folder of the source.

property ExShellFolder.PartialCheck as Boolean

Checks if the source is partial-checked or not.

Type	Description
Boolean	A boolean expression that indicates whether the folder is partially checked.

Use the PartialCheck property to determine whether the folder is partially checked. The Check property determines whether the folder is checked or unchecked. Use the HasCheckBoxes property to assign a check box to each folder in the control's list. Use the PartialCheck property to enable partial check feature. This way, if a folder is checked, it's parent folder is partial checked, if it contains un-checked folders, or full checked, if all child folders are checked.

property ExShellFolder.PathName as String

Retrieves the path of the source.

Type	Description
String	A String expression that specifies the path to the name

The PathName property returns the name of full path.

property ExShellFolder.ShareName as String

Retrieves the share folder name.

Type	Description
String	A String expression that indicates the name of the shared folder.

This read-only property returns the share name of the current folder (as defined in Windows networking). If the folder is not shared, ShareName returns an empty string.

property ExShellFolder.UserData as Variant

Associates an extra data.

Type	Description
Variant	A VARIANT expression that's associated with the current folder.

The UserData property associates an extra data.

ExShellFolders object

The ExShellFolders collection holds a collection of ExShellFolder objects. The Folders or FoldersCheck property retrieves a collection of ExShellFolder objects. The ExShellFolders property supports the following methods and properties.

Name	Description
Add	Not implemented, read-only collection
Count	Returns the number of folders
Item	Returns a folder object
Remove	Read-only collection, method not implemented
RemoveAll	Removes all folders from the collection.

method ExShellFolders.Add (Position as Long)

Not implemented, read-only collection

Type	Description
Position as Long	A long expression that indicates the position where the object is inserted.

Return	Description
Object	An reference to an Object,

property ExShellFolders.Count as Long

Returns the number of folders

Type	Description
Long	A Long expression that indicates the number of elements in the collection.

Read only property providing the number of folders in a in a given folder collection. Use the Item property to access an element in the collection giving its index.

property ExShellFolders.Item (ID as Variant) as ExShellFolder

Returns a folder object

Type	Description
ID as Variant	A long expression
ExShellFolder	An ExShellFolder object being requested.

Individual folders in a Folder collection are accessed by ordinal, the index parameter. Index must always be an integer type, such as Int or Long.

method ExShellFolders.Remove (Item as Variant)

Read-only collection, method not implemented

Type	Description
Item as Variant	A Long expression that specifies the index of the folder being removed from the collection.

Currently, the Remove method is only for internal use.

method ExShellFolders.RemoveAll ()

Removes all folders from the collection.

Type	Description

ExFolderCombo events

The FolderCombo component supports the following events:

Name	Description
NewFolderOpened	Occurs when a new folder is opened.

event NewFolderOpened ()

Occurs when a new folder is opened.

Type	Description

Fired when a new folder is opened by the user. The newly opened folder is found in the OpenedFolder property. The SelectedFolder property indicates the selected folder in the ExFolderView control. The ShellFolder property retrieves the shell folder based on the folder's identifier.

The following VB sample shows how to link a the drop down control with a EXFolderView control:

```
Private Sub ExFolderCombo1_NewFolderOpened()
   ExFolderView1.SelectedFolder = ExFolderCombo1.OpenedFolder
End Sub
```

The following VB sample displays the path of the newly selected folder:

```
Private Sub ExFolderCombo1_NewFolderOpened()
   Debug.Print ExFolderView1.ShellFolder(ExFolderCombo1.OpenedFolder).PathName
End Sub
```

The following C# sample shows how to link a the drop down control with a EXFolderView control:

```
private void axExFolderCombo1_NewFolderOpened(object sender, EventArgs e)
{
   axExFolderView1.SelectedFolder = axExFolderCombo1.OpenedFolder;
}
```

The following VB.NET sample shows how to link a the drop down control with a EXFolderView control:

```
Private Sub AxExFolderCombo1_NewFolderOpened(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles AxExFolderCombo1.NewFolderOpened
   AxExFolderView1.SelectedFolder = AxExFolderCombo1.OpenedFolder
End Sub
```

The following VC sample shows how to link a the drop down control with a EXFolderView control:

```
void OnNewFolderOpenedFoldercombo1()
{
   if ( IsWindow( m_folderCombo.m_hWnd ) && IsWindow( m_folderView.m_hWnd ) )
      m_folderView.SetSelectedFolder( m_folderCombo.GetOpenedFolder() );
}
```

The following VFP sample shows how to link a the drop down control with a EXFolderView control:

```
thisform.ExFolderView1.SelectedFolder = thisform.ExFolderCombo1.OpenedFolder
```

ExFolderView events

The control supports the following events:

Name	Description
AfterCheck	Fired after the user checks a folder.
AfterCollapse	Fired after folder has collapsed.
AfterExpand	Fired after a folder has expanded.
AfterSelChanged	Fired after selection has changed.
AfterShellMenuCommand	Fired after the control executes the selected menu item from the folder's context menu list.
BeforeCollapse	Fired before a folder is about to collapse.
BeforeExpand	Fired before folder is about to be expanded.
BeforeSelChanged	Fired before a folder is selected.
BeforeShellMenuCommand	Fired before a context-menu item is executed.
Click	Occurs when the user presses and then releases the left mouse button over the control.
DblClick	Occurs when the user dblclk the left mouse button over an object.
DropFiles	The user drags a collection of files over the control.
DropQueryEffect	Fired while the user is dragging a folder.
FolderUpdate	This is fired when one of the logical drive was change the state.
IncludeFolder	Occurs when the user includes folders to the control.
KeyDown	Occurs when the user presses a key while an object has the focus.
KeyPress	Occurs when the user presses and releases an ANSI key.
KeyUp	Occurs when the user releases a key while an object has the focus.
MouseDown	Fired when the user release one of the buttons mouse.
MouseMove	Fired when the user move the mouse over the ExFolderView control.
MouseUp	Fired when user release the mouse over the ExFolderView control.
QueryContextMenu	Fired when the context menu is about to be active.

event AfterCheck (Folder as ExShellFolder)

Fired after the user checks a folder.

Type	Description
Folder as ExShellFolder	An object reference to the Folder being checked.

The AfterCheck event notifies your application when a folder is checked or unchecked. Even if the folder was partially checked this event is fired. Use the HasCheckBoxes property to display a checkbox for each folder. Use the PartialCheck property to enable partial check feature. This way, if a folder is checked, it's parent folder is partial checked, if it contains un-checked folders, or full checked, if all child folders are checked. Use the FoldersCheck property to get the collection of checked folders, at run-time.

Syntax for AfterCheck event, **/NET** version, on:

C#

```
private void AfterCheck(object sender,exontrol.EXFOLDERVIEWLib.ExShellFolder Folder)
{
}
```

VB

```
Private Sub AfterCheck(ByVal sender As System.Object,ByVal Folder As exontrol.EXFOLDERVIEWLib.ExShellFolder) Handles AfterCheck
End Sub
```

Syntax for AfterCheck event, **/COM** version, on:

C#

```
private void AfterCheck(object sender, AxEXFOLDERVIEWLib._IExFolderViewEvents_AfterCheckEvent e)
{
}
```

C++

```
void OnAfterCheck(LPDISPATCH Folder)
{
}
```

C++ Builder

```
void __fastcall AfterCheck(TObject *Sender,Exfolderviewlib_tlb::IExShellFolder *Folder)
{
}
```

Delphi

```
procedure AfterCheck(ASender: TObject; Folder : IExShellFolder);
begin
end;
```

Delphi 8 (.NET only)

```
procedure AfterCheck(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_AfterCheckEvent);
begin
end;
```

Powe…

```
begin event AfterCheck(oleobject Folder)
end event AfterCheck
```

VB.NET

```
Private Sub AfterCheck(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_AfterCheckEvent) Handles AfterCheck
End Sub
```

VB6

```
Private Sub AfterCheck(ByVal Folder As EXFOLDERVIEWLibCtl.IExShellFolder)
End Sub
```

VBA

```
Private Sub AfterCheck(ByVal Folder As Object)
End Sub
```

VFP

```
LPARAMETERS Folder

```

Xbas…

```
PROCEDURE OnAfterCheck(oExFolderView,Folder)
RETURN
```

Syntax for AfterCheck event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="AfterCheck(Folder)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function AfterCheck(Folder)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComAfterCheck Variant llFolder
    Forward Send OnComAfterCheck llFolder
End_Procedure
```

Visual Objects

```
METHOD OCX_AfterCheck(Folder) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AfterCheck(COM _Folder)
{
}
```

XBasic

```
function AfterCheck as v (Folder as OLE::Exontrol.FolderView.1::IExShellFolder)
end function
```

dBASE

```
function nativeObject_AfterCheck(Folder)
return
```

event AfterCollapse (SelFolder as ExShellFolder)

Fired after folder has collapsed.

Type	Description
SelFolder as ExShellFolder	A Folder being collapsed.

When user double clicks on a folder, or on a '+' button next to a folder, it expands. Or, if it was already expanded, it collapses. When a folder collapses, two events are fired. BeforeCollapse is fired **before** ExFolderView changes its structure, so additional functions can be executed. The AfterCollapse event is fired **after** the structure has changed. Usually, expanding and collapsing folders will result in more or fewer folders visible after this event, so if some additional code is required it should be put inside this event.

Syntax for AfterCollapse event, **/NET** version, on:

C#

```
private void AfterCollapse(object
sender,exontrol.EXFOLDERVIEWLib.ExShellFolder SelFolder)
{
}
```

VB

```
Private Sub AfterCollapse(ByVal sender As System.Object,ByVal SelFolder As
exontrol.EXFOLDERVIEWLib.ExShellFolder) Handles AfterCollapse
End Sub
```

Syntax for AfterCollapse event, **/COM** version, on:

C#

```
private void AfterCollapse(object sender,
AxEXFOLDERVIEWLib._IExFolderViewEvents_AfterCollapseEvent e)
{
}
```

C++

```
void OnAfterCollapse(LPDISPATCH SelFolder)
{
}
```

C++ Builder

```
void __fastcall AfterCollapse(TObject *Sender,Exfolderviewlib_tlb::IExShellFolder
*SelFolder)
{
}
```

Delphi
```
procedure AfterCollapse(ASender: TObject; SelFolder : IExShellFolder);
begin
end;
```

Delphi 8 (.NET only)
```
procedure AfterCollapse(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_AfterCollapseEvent);
begin
end;
```

Powe…
```
begin event AfterCollapse(oleobject SelFolder)
end event AfterCollapse
```

VB.NET
```
Private Sub AfterCollapse(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_AfterCollapseEvent) Handles
AfterCollapse
End Sub
```

VB6
```
Private Sub AfterCollapse(ByVal SelFolder As EXFOLDERVIEWLibCtl.IExShellFolder)
End Sub
```

VBA
```
Private Sub AfterCollapse(ByVal SelFolder As Object)
End Sub
```

VFP
```
LPARAMETERS SelFolder

```

Xbas…
```
PROCEDURE OnAfterCollapse(oExFolderView,SelFolder)
RETURN
```

Syntax for AfterCollapse event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="AfterCollapse(SelFolder)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function AfterCollapse(SelFolder)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComAfterCollapse Variant llSelFolder
    Forward Send OnComAfterCollapse llSelFolder
End_Procedure
```

Visual Objects

```
METHOD OCX_AfterCollapse(SelFolder) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AfterCollapse(COM _SelFolder)
{
}
```

XBasic

```
function AfterCollapse as v (SelFolder as
OLE::Exontrol.FolderView.1::IExShellFolder)
end function
```

dBASE

```
function nativeObject_AfterCollapse(SelFolder)
return
```

event AfterExpand (SelFolder as ExShellFolder)

Fired after a folder has expanded.

Type	Description
SelFolder as ExShellFolder	A Folder object being expanded.

When user double clicks on a folder, or on a 'plus' button next to a folder, it expands. If it was already expanded, it collapses. When a folder expands, two events are fired. The BeforeExpand is fired **before** ExFolderView changes its structure, so additional functions can be executed. The AfterExpand event is fired **after** the structure has changed. Usually, expanding and collapsing folders will result in more or fewer folders visible after this event, so if some additional code is required it should be put inside this event body.

Syntax for AfterExpand event, **/NET** version, on:

C#

```
private void AfterExpand(object sender,exontrol.EXFOLDERVIEWLib.ExShellFolder SelFolder)
{
}
```

VB

```
Private Sub AfterExpand(ByVal sender As System.Object,ByVal SelFolder As exontrol.EXFOLDERVIEWLib.ExShellFolder) Handles AfterExpand
End Sub
```

Syntax for AfterExpand event, **/COM** version, on:

C#

```
private void AfterExpand(object sender, AxEXFOLDERVIEWLib._IExFolderViewEvents_AfterExpandEvent e)
{
}
```

C++

```
void OnAfterExpand(LPDISPATCH SelFolder)
{
}
```

C++ Builder

```
void __fastcall AfterExpand(TObject *Sender,Exfolderviewlib_tlb::IExShellFolder *SelFolder)
{
}
```

Delphi
```
procedure AfterExpand(ASender: TObject; SelFolder : IExShellFolder);
begin
end;
```

Delphi 8 (.NET only)
```
procedure AfterExpand(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_AfterExpandEvent);
begin
end;
```

Powe…
```
begin event AfterExpand(oleobject SelFolder)
end event AfterExpand
```

VB.NET
```
Private Sub AfterExpand(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_AfterExpandEvent) Handles
AfterExpand
End Sub
```

VB6
```
Private Sub AfterExpand(ByVal SelFolder As EXFOLDERVIEWLibCtl.IExShellFolder)
End Sub
```

VBA
```
Private Sub AfterExpand(ByVal SelFolder As Object)
End Sub
```

VFP
```
LPARAMETERS SelFolder

```

Xbas…
```
PROCEDURE OnAfterExpand(oExFolderView,SelFolder)
RETURN
```

Syntax for AfterExpand event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="AfterExpand(SelFolder)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function AfterExpand(SelFolder)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComAfterExpand Variant llSelFolder
    Forward Send OnComAfterExpand llSelFolder
End_Procedure
```

Visual Objects

```
METHOD OCX_AfterExpand(SelFolder) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AfterExpand(COM _SelFolder)
{
}
```

XBasic

```
function AfterExpand as v (SelFolder as OLE::Exontrol.FolderView.1::IExShellFolder)
end function
```

dBASE

```
function nativeObject_AfterExpand(SelFolder)
return
```

event AfterSelChanged (OldFolder as ExShellFolder, NewFolder as ExShellFolder)

Fired after selection has changed.

Type	Description
OldFolder as ExShellFolder	A Folder object being unselected.
NewFolder as ExShellFolder	A Folder object being selected.

During run-time user clicks inside ExFolderView borders will result in a change to the current selection. There is always at least one folder selected. By clicking on some other folder, selection changes to newly clicked folder. After that happens, this event is fired. Note that this event **does not** occur if some change happens to currently selected folder (like its name, for instance), but only when some other folder is selected. The SelectedFolder property indicates the selected folder, at runtime. The control fires the BeforeSelChanged event to notify the control that the selection is about to be changed.

Syntax for AfterSelChanged event, **/NET** version, on:

C#
```
private void AfterSelChanged(object
sender,exontrol.EXFOLDERVIEWLib.ExShellFolder
OldFolder,exontrol.EXFOLDERVIEWLib.ExShellFolder NewFolder)
{
}
```

VB
```
Private Sub AfterSelChanged(ByVal sender As System.Object,ByVal OldFolder As
exontrol.EXFOLDERVIEWLib.ExShellFolder,ByVal NewFolder As
exontrol.EXFOLDERVIEWLib.ExShellFolder) Handles AfterSelChanged
End Sub
```

Syntax for AfterSelChanged event, **/COM** version, on:

C#
```
private void AfterSelChanged(object sender,
AxEXFOLDERVIEWLib._IExFolderViewEvents_AfterSelChangedEvent e)
{
}
```

C++
```
void OnAfterSelChanged(LPDISPATCH OldFolder,LPDISPATCH NewFolder)
{
}
```

C++ Builder

```
void __fastcall AfterSelChanged(TObject
*Sender,Exfolderviewlib_tlb::IExShellFolder
*OldFolder,Exfolderviewlib_tlb::IExShellFolder *NewFolder)
{
}
```

Delphi

```
procedure AfterSelChanged(ASender: TObject; OldFolder :
IExShellFolder;NewFolder : IExShellFolder);
begin
end;
```

Delphi 8 (.NET only)

```
procedure AfterSelChanged(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_AfterSelChangedEvent);
begin
end;
```

Powe…

```
begin event AfterSelChanged(oleobject OldFolder,oleobject NewFolder)
end event AfterSelChanged
```

VB.NET

```
Private Sub AfterSelChanged(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_AfterSelChangedEvent) Handles
AfterSelChanged
End Sub
```

VB6

```
Private Sub AfterSelChanged(ByVal OldFolder As
EXFOLDERVIEWLibCtl.IExShellFolder,ByVal NewFolder As
EXFOLDERVIEWLibCtl.IExShellFolder)
End Sub
```

VBA

```
Private Sub AfterSelChanged(ByVal OldFolder As Object,ByVal NewFolder As
Object)
End Sub
```

VFP

```
LPARAMETERS OldFolder,NewFolder

```

Xbas…

```
PROCEDURE OnAfterSelChanged(oExFolderView,OldFolder,NewFolder)
RETURN
```

Syntax for AfterSelChanged event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="AfterSelChanged(OldFolder,NewFolder)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function AfterSelChanged(OldFolder,NewFolder)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComAfterSelChanged Variant llOldFolder Variant llNewFolder
    Forward Send OnComAfterSelChanged llOldFolder llNewFolder
End_Procedure
```

Visual Objects
```
METHOD OCX_AfterSelChanged(OldFolder,NewFolder) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_AfterSelChanged(COM _OldFolder,COM _NewFolder)
{
}
```

XBasic
```
function AfterSelChanged as v (OldFolder as
OLE::Exontrol.FolderView.1::IExShellFolder,NewFolder as
OLE::Exontrol.FolderView.1::IExShellFolder)
end function
```

dBASE
```
function nativeObject_AfterSelChanged(OldFolder,NewFolder)
return
```

The following VB sample changes the drop down opened folder when the user changes the selection in the ExFolderView control:

```
Private Sub ExFolderView1_AfterSelChanged(ByVal OldFolder As
EXFOLDERVIEWLibCtl.IExShellFolder, ByVal NewFolder As
EXFOLDERVIEWLibCtl.IExShellFolder)
    ExFolderCombo1.OpenedFolder = ExFolderView1.SelectedFolder
```

```
End Sub
```

The following VB.NET sample changes the drop down opened folder when the user changes the selection in the ExFolderView control:

```
Private Sub AxExFolderView1_AfterSelChanged(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_AfterSelChangedEvent) Handles
AxExFolderView1.AfterSelChanged
    AxExFolderCombo1.OpenedFolder = AxExFolderView1.SelectedFolder
End Sub
```

The following VC sample changes the drop down opened folder when the user changes the selection in the ExFolderView control:

```
void OnAfterSelChangedFolderview1(LPDISPATCH OldFolder, LPDISPATCH NewFolder)
{
    if ( IsWindow( m_folderCombo.m_hWnd ) && IsWindow( m_folderView.m_hWnd ) )
        m_folderCombo.SetOpenedFolder( m_folderView.GetSelectedFolder() );
}
```

The following C# sample changes the drop down opened folder when the user changes the selection in the ExFolderView control:

```
private void axExFolderView1_AfterSelChanged(object sender,
AxEXFOLDERVIEWLib._IExFolderViewEvents_AfterSelChangedEvent e)
{
    axExFolderCombo1.OpenedFolder = axExFolderView1.SelectedFolder;
}
```

The following VFP sample changes the drop down opened folder when the user changes the selection in the ExFolderView control:

```
*** ActiveX Control Event ***
LPARAMETERS oldfolder, newfolder

thisform.ExFolderCombo1.OpenedFolder = thisform.ExFolderView1.SelectedFolder
```

event AfterShellMenuCommand ()

Fired after the control executes the selected menu item from the folder's context menu list.

Type	Description

Right clicking on a folder results in showing the folder's context menu. If some item from this menu is selected, the control fires a pair of events to notify your application about this event. Before the menu item is executed, BeforeShellMenuCommand event is fired. After the menu item is executed, this event is fired. Please note that Windows OS is a multitasking environment. That means that execution of menu item does not mean that this event is fired after the executed program (or command) has finished. Rather, immediately after execution of the menu item this event is fired. The control provides a drop down context menu, if the EnableShellMenu property is True.

Syntax for AfterShellMenuCommand event, **/NET** version, on:

C#
```
private void AfterShellMenuCommand(object sender)
{
}
```

VB
```
Private Sub AfterShellMenuCommand(ByVal sender As System.Object) Handles
AfterShellMenuCommand
End Sub
```

Syntax for AfterShellMenuCommand event, **/COM** version, on:

C#
```
private void AfterShellMenuCommand(object sender, EventArgs e)
{
}
```

C++
```
void OnAfterShellMenuCommand()
{
}
```

C++ Builder
```
void __fastcall AfterShellMenuCommand(TObject *Sender)
{
}
```

Delphi
```
procedure AfterShellMenuCommand(ASender: TObject; );
begin
```

```
end;
```

Delphi 8 (.NET only)

```
procedure AfterShellMenuCommand(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event AfterShellMenuCommand()
end event AfterShellMenuCommand
```

VB.NET

```
Private Sub AfterShellMenuCommand(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles AfterShellMenuCommand
End Sub
```

VB6

```
Private Sub AfterShellMenuCommand()
End Sub
```

VBA

```
Private Sub AfterShellMenuCommand()
End Sub
```

VFP

```
LPARAMETERS nop

```

Xbas…

```
PROCEDURE OnAfterShellMenuCommand(oExFolderView)
RETURN
```

Syntax for AfterShellMenuCommand event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="AfterShellMenuCommand()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function AfterShellMenuCommand()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComAfterShellMenuCommand
    Forward Send OnComAfterShellMenuCommand
End_Procedure
```

Visual Objects

```
METHOD OCX_AfterShellMenuCommand() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AfterShellMenuCommand()
{
}
```

XBasic

```
function AfterShellMenuCommand as v ()
end function
```

dBASE

```
function nativeObject_AfterShellMenuCommand()
return
```

event BeforeCollapse (SelFolder as ExShellFolder, Cancel as Boolean)

Fired before a folder is about to collapse.

Type	Description
SelFolder as ExShellFolder	A Folder being collapsed
Cancel as Boolean	A Boolen expression that indicates whether the operation is canceled or not.

When the user double clicks on a folder, or on a 'plus' button next to folder, it expands. If it was already expanded, it collapses. When the folder collapses, two events are fired. BeforeCollapse is fired **before** ExFolderView changes its structure, so some additional functions can be executed. The AfterCollapse event is fired **after** the structure has changed. Usually, expanding and collapsing folders will result in more or fewer folders being visible after this event, so if some additional code is required it should be put inside this event body.

Syntax for BeforeCollapse event, **/NET** version, on:

C#
```
private void BeforeCollapse(object
sender,exontrol.EXFOLDERVIEWLib.ExShellFolder SelFolder,ref bool Cancel)
{
}
```

VB
```
Private Sub BeforeCollapse(ByVal sender As System.Object,ByVal SelFolder As
exontrol.EXFOLDERVIEWLib.ExShellFolder,ByRef Cancel As Boolean) Handles
BeforeCollapse
End Sub
```

Syntax for BeforeCollapse event, **/COM** version, on:

C#
```
private void BeforeCollapse(object sender,
AxEXFOLDERVIEWLib._IExFolderViewEvents_BeforeCollapseEvent e)
{
}
```

C++
```
void OnBeforeCollapse(LPDISPATCH SelFolder,BOOL FAR* Cancel)
{
}
```

C++ Builder

```
void __fastcall BeforeCollapse(TObject *Sender,Exfolderviewlib_tlb::IExShellFolder
*SelFolder,VARIANT_BOOL * Cancel)
{
}
```

Delphi

```
procedure BeforeCollapse(ASender: TObject; SelFolder : IExShellFolder;var Cancel :
WordBool);
begin
end;
```

Delphi 8 (.NET only)

```
procedure BeforeCollapse(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_BeforeCollapseEvent);
begin
end;
```

Powe…

```
begin event BeforeCollapse(oleobject SelFolder,boolean Cancel)
end event BeforeCollapse
```

VB.NET

```
Private Sub BeforeCollapse(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_BeforeCollapseEvent) Handles
BeforeCollapse
End Sub
```

VB6

```
Private Sub BeforeCollapse(ByVal SelFolder As
EXFOLDERVIEWLibCtl.IExShellFolder,Cancel As Boolean)
End Sub
```

VBA

```
Private Sub BeforeCollapse(ByVal SelFolder As Object,Cancel As Boolean)
End Sub
```

VFP

```
LPARAMETERS SelFolder,Cancel

```

Xbas…

```
PROCEDURE OnBeforeCollapse(oExFolderView,SelFolder,Cancel)
RETURN
```

Syntax for BeforeCollapse event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="BeforeCollapse(SelFolder,Cancel)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function BeforeCollapse(SelFolder,Cancel)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComBeforeCollapse Variant llSelFolder Boolean llCancel
    Forward Send OnComBeforeCollapse llSelFolder llCancel
End_Procedure
```

Visual Objects
```
METHOD OCX_BeforeCollapse(SelFolder,Cancel) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_BeforeCollapse(COM _SelFolder,COMVariant /*bool*/ _Cancel)
{
}
```

XBasic
```
function BeforeCollapse as v (SelFolder as
OLE::Exontrol.FolderView.1::IExShellFolder,Cancel as L)
end function
```

dBASE
```
function nativeObject_BeforeCollapse(SelFolder,Cancel)
return
```

event BeforeExpand (SelFolder as ExShellFolder, Cancel as Boolean)

Fired before folder is about to be expanded.

Type	Description
SelFolder as ExShellFolder	A Folder being expanded.
Cancel as Boolean	A Boolean expression that indicates whether the operation is executed or canceled.

When the user double clicks on a folder, or on a 'plus' button next to folder, it expands. If it was already expanded, it collapses. When a folder expands, two events are fired. The BeforeExpand is fired **before** ExFolderView changes its structure, so additional functions can be executed. The AfterExpand event is fired **after** the structure has changed. Usually, expanding and collapsing folders will result in more or less folders visible after this event, so if additional code is required it should be put inside this event body.

Syntax for BeforeExpand event, **/NET** version, on:

C#
```
private void BeforeExpand(object
sender,exontrol.EXFOLDERVIEWLib.ExShellFolder SelFolder,ref bool Cancel)
{
}
```

VB
```
Private Sub BeforeExpand(ByVal sender As System.Object,ByVal SelFolder As
exontrol.EXFOLDERVIEWLib.ExShellFolder,ByRef Cancel As Boolean) Handles
BeforeExpand
End Sub
```

Syntax for BeforeExpand event, **/COM** version, on:

C#
```
private void BeforeExpand(object sender,
AxEXFOLDERVIEWLib._IExFolderViewEvents_BeforeExpandEvent e)
{
}
```

C++
```
void OnBeforeExpand(LPDISPATCH SelFolder,BOOL FAR* Cancel)
{
}
```

C++ Builder
```
void __fastcall BeforeExpand(TObject *Sender,Exfolderviewlib_tlb::IExShellFolder
*SelFolder,VARIANT_BOOL * Cancel)
```

```
{
}
```

Delphi

```
procedure BeforeExpand(ASender: TObject; SelFolder : IExShellFolder;var Cancel :
WordBool);
begin
end;
```

Delphi 8 (.NET only)

```
procedure BeforeExpand(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_BeforeExpandEvent);
begin
end;
```

Powe…

```
begin event BeforeExpand(oleobject SelFolder,boolean Cancel)
end event BeforeExpand
```

VB.NET

```
Private Sub BeforeExpand(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_BeforeExpandEvent) Handles
BeforeExpand
End Sub
```

VB6

```
Private Sub BeforeExpand(ByVal SelFolder As
EXFOLDERVIEWLibCtl.IExShellFolder,Cancel As Boolean)
End Sub
```

VBA

```
Private Sub BeforeExpand(ByVal SelFolder As Object,Cancel As Boolean)
End Sub
```

VFP

```
LPARAMETERS SelFolder,Cancel

```

Xbas…

```
PROCEDURE OnBeforeExpand(oExFolderView,SelFolder,Cancel)
RETURN
```

Syntax for BeforeExpand event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="BeforeExpand(SelFolder,Cancel)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function BeforeExpand(SelFolder,Cancel)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComBeforeExpand Variant llSelFolder Boolean llCancel
    Forward Send OnComBeforeExpand llSelFolder llCancel
End_Procedure
```

Visual Objects

```
METHOD OCX_BeforeExpand(SelFolder,Cancel) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_BeforeExpand(COM _SelFolder,COMVariant /*bool*/ _Cancel)
{
}
```

XBasic

```
function BeforeExpand as v (SelFolder as
OLE::Exontrol.FolderView.1::IExShellFolder,Cancel as L)
end function
```

dBASE

```
function nativeObject_BeforeExpand(SelFolder,Cancel)
return
```

event BeforeSelChanged (OldFolder as ExShellFolder, NewFolder as ExShellFolder, Cancel as Boolean)

Fired before a folder is selected.

Type	Description
OldFolder as ExShellFolder	A Folder being un-selected.
NewFolder as ExShellFolder	A Folder being selected.
Cancel as Boolean	A Boolen expression that indicates whether the operation is executed or canceled.

This event is fired before the current selection is changed. Setting the Cancel parameter to True prevents the selection from being changed. The NewFolder parameter lets you know which folder will become selected if you do not cancel the selection change and the OldFolder parameter lets you know which folder is currently selected. The SelectedFolder property indicates the selected folder, at runtime. The control fires the AfterSelChanged event after the user changes the selection.

Syntax for BeforeSelChanged event, **/NET** version, on:

C#
```
private void BeforeSelChanged(object
sender,exontrol.EXFOLDERVIEWLib.ExShellFolder
OldFolder,exontrol.EXFOLDERVIEWLib.ExShellFolder NewFolder,ref bool Cancel)
{
}
```

VB
```
Private Sub BeforeSelChanged(ByVal sender As System.Object,ByVal OldFolder As
exontrol.EXFOLDERVIEWLib.ExShellFolder,ByVal NewFolder As
exontrol.EXFOLDERVIEWLib.ExShellFolder,ByRef Cancel As Boolean) Handles
BeforeSelChanged
End Sub
```

Syntax for BeforeSelChanged event, **/COM** version, on:

C#
```
private void BeforeSelChanged(object sender,
AxEXFOLDERVIEWLib._IExFolderViewEvents_BeforeSelChangedEvent e)
{
}
```

C++
```
void OnBeforeSelChanged(LPDISPATCH OldFolder,LPDISPATCH NewFolder,BOOL
```

```
FAR* Cancel)
{
}
```

C++ Builder

```
void __fastcall BeforeSelChanged(TObject
*Sender,Exfolderviewlib_tlb::IExShellFolder
*OldFolder,Exfolderviewlib_tlb::IExShellFolder *NewFolder,VARIANT_BOOL *
Cancel)
{
}
```

Delphi

```
procedure BeforeSelChanged(ASender: TObject; OldFolder :
IExShellFolder;NewFolder : IExShellFolder;var Cancel : WordBool);
begin
end;
```

Delphi 8 (.NET only)

```
procedure BeforeSelChanged(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_BeforeSelChangedEvent);
begin
end;
```

Powe…

```
begin event BeforeSelChanged(oleobject OldFolder,oleobject NewFolder,boolean
Cancel)
end event BeforeSelChanged
```

VB.NET

```
Private Sub BeforeSelChanged(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_BeforeSelChangedEvent) Handles
BeforeSelChanged
End Sub
```

VB6

```
Private Sub BeforeSelChanged(ByVal OldFolder As
EXFOLDERVIEWLibCtl.IExShellFolder,ByVal NewFolder As
EXFOLDERVIEWLibCtl.IExShellFolder,Cancel As Boolean)
End Sub
```

VBA

```
Private Sub BeforeSelChanged(ByVal OldFolder As Object,ByVal NewFolder As
Object,Cancel As Boolean)
End Sub
```

VFP

```
LPARAMETERS OldFolder,NewFolder,Cancel
```

Xbas…

```
PROCEDURE OnBeforeSelChanged(oExFolderView,OldFolder,NewFolder,Cancel)
RETURN
```

Syntax for BeforeSelChanged event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="BeforeSelChanged(OldFolder,NewFolder,Cancel)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function BeforeSelChanged(OldFolder,NewFolder,Cancel)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComBeforeSelChanged Variant llOldFolder Variant llNewFolder
Boolean llCancel
    Forward Send OnComBeforeSelChanged llOldFolder llNewFolder llCancel
End_Procedure
```

Visual Objects

```
METHOD OCX_BeforeSelChanged(OldFolder,NewFolder,Cancel) CLASS
MainDialog
RETURN NIL
```

X++

```
void onEvent_BeforeSelChanged(COM _OldFolder,COM _NewFolder,COMVariant
/*bool*/ _Cancel)
{
}
```

XBasic

```
function BeforeSelChanged as v (OldFolder as
OLE::Exontrol.FolderView.1::IExShellFolder,NewFolder as
OLE::Exontrol.FolderView.1::IExShellFolder,Cancel as L)
end function
```

dBASE

```
function nativeObject_BeforeSelChanged(OldFolder,NewFolder,Cancel)
return
```

event BeforeShellMenuCommand (Command as String, ID as Long, Cancel as Boolean)

Fired before a context-menu item is executed.

Type	Description
Command as String	A string expression that specifies context menu item selected.
ID as Long	A long integer that represents the unique ID of selected menu item.
Cancel as Boolean	A boolean expression that, if set to True, execution of selected context menu item will be denied. If set to False, menu item will be executed.

Right clicking on a folder results in the folder's context menu being displayed. If some item from this menu is selected, the control fires a pair of events to notify you. Before the menu item is executed, this event is fired. After the menu item is executed, AfterShellMenuCommand event is fired.

When this event is fired, you may deny execution of the selected menu item. Command variable holds name of selected menu item, and nID variable holds its unique index inside menu-items collection. Depending on user's choice, setting Cancel value to True will deny execution of selected menu item.

When checking Command's value, please keep in mind that some menu items have underscored letters. For example, 'Open' menu item usually have letter 'O' with underscore, which means that pressing ALT+O will result in executing of 'Open' command. Such underscored letters are stored internally in Windows OS by prefixing such letter with a '&' sign. So, if you check Command, don't forget to include this sign into string you will check Command with.

Following following code will not work correctly:

```
If Command = "Open" Then MsgBox "Folder will be opened"
```

because '&Open' and 'Open' is not the same string. Rather, use this line (tested in VB):

```
If Command = "&Open" Then MsgBox "Folder will be opened"
```

Syntax for BeforeShellMenuCommand event, **/NET** version, on:

C#

```
private void BeforeShellMenuCommand(object sender,string Command,int ID,ref
bool Cancel)
{
```

```
}
```

VB

```
Private Sub BeforeShellMenuCommand(ByVal sender As System.Object,ByVal
Command As String,ByVal ID As Integer,ByRef Cancel As Boolean) Handles
BeforeShellMenuCommand
End Sub
```

Syntax for BeforeShellMenuCommand event, **/COM** version, on:

C#

```
private void BeforeShellMenuCommand(object sender,
AxEXFOLDERVIEWLib._IExFolderViewEvents_BeforeShellMenuCommandEvent e)
{
}
```

C++

```
void OnBeforeShellMenuCommand(LPCTSTR Command,long ID,BOOL FAR*
Cancel)
{
}
```

C++ Builder

```
void __fastcall BeforeShellMenuCommand(TObject *Sender,BSTR Command,long
ID,VARIANT_BOOL * Cancel)
{
}
```

Delphi

```
procedure BeforeShellMenuCommand(ASender: TObject; Command :
WideString;ID : Integer;var Cancel : WordBool);
begin
end;
```

Delphi 8 (.NET only)

```
procedure BeforeShellMenuCommand(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_BeforeShellMenuCommandEvent);
begin
end;
```

Powe…

```
begin event BeforeShellMenuCommand(string Command,long ID,boolean Cancel)
end event BeforeShellMenuCommand
```

VB.NET

```
Private Sub BeforeShellMenuCommand(ByVal sender As System.Object, ByVal e As
```

```
AxEXFOLDERVIEWLib._IExFolderViewEvents_BeforeShellMenuCommandEvent)
Handles BeforeShellMenuCommand
End Sub
```

VB6

```
Private Sub BeforeShellMenuCommand(ByVal Command As String,ByVal ID As
Long,Cancel As Boolean)
End Sub
```

VBA

```
Private Sub BeforeShellMenuCommand(ByVal Command As String,ByVal ID As
Long,Cancel As Boolean)
End Sub
```

VFP

```
LPARAMETERS Command,ID,Cancel
```

Xbas…

```
PROCEDURE OnBeforeShellMenuCommand(oExFolderView,Command,ID,Cancel)
RETURN
```

Syntax for BeforeShellMenuCommand event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="BeforeShellMenuCommand(Command,ID,Cancel)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function BeforeShellMenuCommand(Command,ID,Cancel)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComBeforeShellMenuCommand String llCommand Integer llID
Boolean llCancel
    Forward Send OnComBeforeShellMenuCommand llCommand llID llCancel
End_Procedure
```

Visual Objects

```
METHOD OCX_BeforeShellMenuCommand(Command,ID,Cancel) CLASS
MainDialog
RETURN NIL
```

X++

```
void onEvent_BeforeShellMenuCommand(str _Command,int _ID,COMVariant
/*bool*/ _Cancel)
{
}
```

XBasic

```
function BeforeShellMenuCommand as v (Command as C,ID as N,Cancel as L)
end function
```

dBASE

```
function nativeObject_BeforeShellMenuCommand(Command,ID,Cancel)
return
```

event Click ()

Occurs when the user presses and then releases the left mouse button over the control

Type	Description

The Click event is fired when the user releases the left mouse button over the control. Use a MouseDown or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Syntax for Click event, **/NET** version, on:

C#

```
private void Click(object sender)
{
}
```

VB

```
Private Sub Click(ByVal sender As System.Object) Handles Click
End Sub
```

Syntax for Click event, **/COM** version, on:

C#

```
private void ClickEvent(object sender, EventArgs e)
{
}
```

C++

```
void OnClick()
{
}
```

C++ Builder

```
void __fastcall Click(TObject *Sender)
{
}
```

Delphi

```
procedure Click(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)

```
procedure ClickEvent(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…

```
begin event Click()
end event Click
```

VB.NET

```
Private Sub ClickEvent(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ClickEvent
End Sub
```

VB6

```
Private Sub Click()
End Sub
```

VBA

```
Private Sub Click()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnClick(oExFolderView)
RETURN
```

Syntax for Click event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="Click()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function Click()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComClick
    Forward Send OnComClick
```

```
End_Procedure
```

Visual Objects

```
METHOD OCX_Click() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_Click()
{
}
```

XBasic

```
function Click as v ()
end function
```

dBASE

```
function nativeObject_Click()
return
```

event DblClick ()

Occurs when the user dblclk the left mouse button over an object.

Type	Description

The DblClick event is fired when user double clicks the control.

Syntax for DblClick event, **/NET** version, on:

C#
```
private void DblClick(object sender)
{
}
```

VB
```
Private Sub DblClick(ByVal sender As System.Object) Handles DblClick
End Sub
```

Syntax for DblClick event, **/COM** version, on:

C#
```
private void DblClick(object sender, EventArgs e)
{
}
```

C++
```
void OnDblClick()
{
}
```

C++ Builder
```
void __fastcall DblClick(TObject *Sender)
{
}
```

Delphi
```
procedure DblClick(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)
```
procedure DblClick(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…
```
begin event DblClick()
end event DblClick
```

VB.NET

```
Private Sub DblClick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles DblClick
End Sub
```

VB6

```
Private Sub DblClick()
End Sub
```

VBA

```
Private Sub DblClick()
End Sub
```

VFP

```
LPARAMETERS nop

```

Xbas…

```
PROCEDURE OnDblClick(oExFolderView)
RETURN
```

Syntax for DblClick event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="DblClick()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function DblClick()
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComDblClick
    Forward Send OnComDblClick
End_Procedure
```

Visual Objects

```
METHOD OCX_DblClick() CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_DblClick()
{
}
```

XBasic

```
function DblClick as v ()
end function
```

dBASE

```
function nativeObject_DblClick()
return
```

event DropFiles (Folder as ExShellFolder, Effect as Long)

The user drags a collection of files over the control.

Type	Description
Folder as ExShellFolder	An object reference to a Folder object.
Effect as Long	A long integer that represents the desired drag-and-drop effect.

This event occurs after the user drags and drops files into the control. Folder is the target of the drag and drop operation. Effect holds the desired effect of the drag and drop. Use the DropFilesCount property to count the files being dropped. Use the DropFilesPathName property to retrieve the path of the dropping folder. The DropFiles event is fired only if the AllowDropFiles property is True.

Syntax for DropFiles event, **/NET** version, on:

C#
```
private void DropFiles(object sender,exontrol.EXFOLDERVIEWLib.ExShellFolder Folder,int Effect)
{
}
```

VB
```
Private Sub DropFiles(ByVal sender As System.Object,ByVal Folder As exontrol.EXFOLDERVIEWLib.ExShellFolder,ByVal Effect As Integer) Handles DropFiles
End Sub
```

Syntax for DropFiles event, **/COM** version, on:

C#
```
private void DropFiles(object sender, AxEXFOLDERVIEWLib._IExFolderViewEvents_DropFilesEvent e)
{
}
```

C++
```
void OnDropFiles(LPDISPATCH Folder,long Effect)
{
}
```

C++ Builder
```
void __fastcall DropFiles(TObject *Sender,Exfolderviewlib_tlb::IExShellFolder *Folder,long Effect)
```

```
{
}
```

Delphi

```
procedure DropFiles(ASender: TObject; Folder : IExShellFolder;Effect : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure DropFiles(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_DropFilesEvent);
begin
end;
```

Powe…

```
begin event DropFiles(oleobject Folder,long Effect)
end event DropFiles
```

VB.NET

```
Private Sub DropFiles(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_DropFilesEvent) Handles DropFiles
End Sub
```

VB6

```
Private Sub DropFiles(ByVal Folder As EXFOLDERVIEWLibCtl.IExShellFolder,ByVal
Effect As Long)
End Sub
```

VBA

```
Private Sub DropFiles(ByVal Folder As Object,ByVal Effect As Long)
End Sub
```

VFP

```
LPARAMETERS Folder,Effect

```

Xbas…

```
PROCEDURE OnDropFiles(oExFolderView,Folder,Effect)
RETURN
```

Syntax for DropFiles event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="DropFiles(Folder,Effect)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

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

```
Function DropFiles(Folder,Effect)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComDropFiles Variant llFolder Integer llEffect
    Forward Send OnComDropFiles llFolder llEffect
End_Procedure
```

Visual Objects

```
METHOD OCX_DropFiles(Folder,Effect) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_DropFiles(COM _Folder,int _Effect)
{
}
```

XBasic

```
function DropFiles as v (Folder as OLE::Exontrol.FolderView.1::IExShellFolder,Effect
as N)
end function
```

dBASE

```
function nativeObject_DropFiles(Folder,Effect)
return
```

Here is a VB sample that lists the files dragged in the Immediate debugger window.

```
Private Sub ExFolderView1_DropFiles(ByVal ExShellFolder As
EXFOLDERVIEWLibCtl.IExShellFolder, ByVal Effect As Long)
  Dim I As Long
  For I = 0 To FolderView1.DropFilesCount - 1
    Debug.Print FolderView1.DropFilesPathName(I)
  Next I
  If (Effect & 1) = 1 Then
    Debug.Print "Copied to " & Folder.PathName
  Else
    Debug.Print "Moved to " & Folder.PathName
  End If
End Sub
```

event DropQueryEffect (Folder as ExShellFolder, KeyState as Long, Effect as Long)

Fired while the user is dragging a folder.

Type	Description
Folder as ExShellFolder	A Folder object being droped.
KeyState as Long	A long integer that represents the current state of certain keys on the keyboard.
Effect as Long	A long integer that represents the desired drag-and-drop effect.

Fires while a drag-and-drop operation is in progress. This event fires just prior to the end of a drag-and-drop operation (i.e., just before the DropFiles event). The *Folder* parameter is the target of the drag-and-drop operation. The KeyState holds the state of keys (such as Shift, Ctrl, Alt, etc.) at the end of the operation. Effect holds the desired effect on the files in the operation. The AllowDropFiles property determines whether or not the control will accept files dragged-and-dropped from another application (such as Explorer).

Syntax for DropQueryEffect event, **/NET** version, on:

C#
```
private void DropQueryEffect(object
sender,exontrol.EXFOLDERVIEWLib.ExShellFolder Folder,int KeyState,ref int Effect)
{
}
```

VB
```
Private Sub DropQueryEffect(ByVal sender As System.Object,ByVal Folder As
exontrol.EXFOLDERVIEWLib.ExShellFolder,ByVal KeyState As Integer,ByRef Effect As
Integer) Handles DropQueryEffect
End Sub
```

Syntax for DropQueryEffect event, **/COM** version, on:

C#
```
private void DropQueryEffect(object sender,
AxEXFOLDERVIEWLib._IExFolderViewEvents_DropQueryEffectEvent e)
{
}
```

C++
```
void OnDropQueryEffect(LPDISPATCH Folder,long KeyState,long FAR* Effect)
{
```

```
}
```

C++ Builder

```
void __fastcall DropQueryEffect(TObject
*Sender,Exfolderviewlib_tlb::IExShellFolder *Folder,long KeyState,long * Effect)
{
}
```

Delphi

```
procedure DropQueryEffect(ASender: TObject; Folder : IExShellFolder;KeyState :
Integer;var Effect : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure DropQueryEffect(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_DropQueryEffectEvent);
begin
end;
```

Powe…

```
begin event DropQueryEffect(oleobject Folder,long KeyState,long Effect)
end event DropQueryEffect
```

VB.NET

```
Private Sub DropQueryEffect(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_DropQueryEffectEvent) Handles
DropQueryEffect
End Sub
```

VB6

```
Private Sub DropQueryEffect(ByVal Folder As
EXFOLDERVIEWLibCtl.IExShellFolder,ByVal KeyState As Long,Effect As Long)
End Sub
```

VBA

```
Private Sub DropQueryEffect(ByVal Folder As Object,ByVal KeyState As Long,Effect
As Long)
End Sub
```

VFP

```
LPARAMETERS Folder,KeyState,Effect

```

Xbas…

```
PROCEDURE OnDropQueryEffect(oExFolderView,Folder,KeyState,Effect)
RETURN
```

Syntax for DropQueryEffect event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="DropQueryEffect(Folder,KeyState,Effect)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function DropQueryEffect(Folder,KeyState,Effect)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComDropQueryEffect Variant llFolder Integer llKeyState Integer
llEffect
    Forward Send OnComDropQueryEffect llFolder llKeyState llEffect
End_Procedure
```

Visual Objects
```
METHOD OCX_DropQueryEffect(Folder,KeyState,Effect) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_DropQueryEffect(COM _Folder,int _KeyState,COMVariant /*long*/
_Effect)
{
}
```

XBasic
```
function DropQueryEffect as v (Folder as
OLE::Exontrol.FolderView.1::IExShellFolder,KeyState as N,Effect as N)
end function
```

dBASE
```
function nativeObject_DropQueryEffect(Folder,KeyState,Effect)
return
```

Here is a VB sample that shows you how to cancel a drag-and-drop operation if the target is the "My Computer" folder. In all other cases (targets), the operation is changed to "copy".

```
Private Sub ExFolderView1_DropQueryEffect(ByVal ExShellFolder As
EXFOLDERVIEWLibCtl.IExShellFolder, ByVal KeyState As Long, Effect As Long)
  If ( Folder.DisplayName = "My Computer" )
```

```
    Effect = 0 ' cancel
  Else
    Effect = 1 ' copy
  End If
End Sub
```

event FolderUpdate ()

This is fired when one of the logical drive was change the state.

Type	Description

This is fired when one of the logical drive changes its state. Let's suppose that we have an Explorer opened. Activate it, and rename a folder. You can see that the ExFolderView control fires this event. So, this event is fired when the user delete, create, changes, ... anything on one of the disks. The event is fired only if the AutoUpdate property is True.

Syntax for FolderUpdate event, **/NET** version, on:

C#
```
private void FolderUpdate(object sender)
{
}
```

VB
```
Private Sub FolderUpdate(ByVal sender As System.Object) Handles FolderUpdate
End Sub
```

Syntax for FolderUpdate event, **/COM** version, on:

C#
```
private void FolderUpdate(object sender, EventArgs e)
{
}
```

C++
```
void OnFolderUpdate()
{
}
```

C++ Builder
```
void __fastcall FolderUpdate(TObject *Sender)
{
}
```

Delphi
```
procedure FolderUpdate(ASender: TObject; );
begin
end;
```

Delphi 8 (.NET only)
```
procedure FolderUpdate(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…
```
begin event FolderUpdate()
end event FolderUpdate
```

VB.NET
```
Private Sub FolderUpdate(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles FolderUpdate
End Sub
```

VB6
```
Private Sub FolderUpdate()
End Sub
```

VBA
```
Private Sub FolderUpdate()
End Sub
```

VFP
```
LPARAMETERS nop
```

Xbas…
```
PROCEDURE OnFolderUpdate(oExFolderView)
RETURN
```

Syntax for FolderUpdate event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="FolderUpdate()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function FolderUpdate()
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComFolderUpdate
    Forward Send OnComFolderUpdate
End_Procedure
```

Visual Objects
```
METHOD OCX_FolderUpdate() CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_FolderUpdate()
{
```

```
}
```

XBasic

```
function FolderUpdate as v ()
end function
```

dBASE

```
function nativeObject_FolderUpdate()
return
```

event IncludeFolder (Folder as ExShellFolder, Include as Boolean)

Occurs when the user includes folders to the control

Type	Description
Folder as ExShellFolder	A Folder object being included in the control's list
Include as Boolean	A Boolen expression that specifies whether the Folder is included or excluded from the control's list.

The IncludeFolder event notifies your application that a new folder is included in the control's list. Use the IncludeFolder event to customize the list of folders being included in your list. The IncludeFolder event is fired only if the IncludeFolder property is True.

Syntax for IncludeFolder event, **/NET** version, on:

C#
```
private void IncludeFolder(object
sender,exontrol.EXFOLDERVIEWLib.ExShellFolder Folder,ref bool Include)
{
}
```

VB
```
Private Sub IncludeFolder(ByVal sender As System.Object,ByVal Folder As
exontrol.EXFOLDERVIEWLib.ExShellFolder,ByRef Include As Boolean) Handles
IncludeFolder
End Sub
```

Syntax for IncludeFolder event, **/COM** version, on:

C#
```
private void IncludeFolder(object sender,
AxEXFOLDERVIEWLib._IExFolderViewEvents_IncludeFolderEvent e)
{
}
```

C++
```
void OnIncludeFolder(LPDISPATCH Folder,BOOL FAR* Include)
{
}
```

C++ Builder
```
void __fastcall IncludeFolder(TObject *Sender,Exfolderviewlib_tlb::IExShellFolder
*Folder,VARIANT_BOOL * Include)
{
}
```

Delphi
```
procedure IncludeFolder(ASender: TObject; Folder : IExShellFolder;var Include : WordBool);
begin
end;
```

Delphi 8 (.NET only)
```
procedure IncludeFolder(sender: System.Object; e: AxEXFOLDERVIEWLib._IExFolderViewEvents_IncludeFolderEvent);
begin
end;
```

Powe…
```
begin event IncludeFolder(oleobject Folder,boolean Include)
end event IncludeFolder
```

VB.NET
```
Private Sub IncludeFolder(ByVal sender As System.Object, ByVal e As AxEXFOLDERVIEWLib._IExFolderViewEvents_IncludeFolderEvent) Handles IncludeFolder
End Sub
```

VB6
```
Private Sub IncludeFolder(ByVal Folder As EXFOLDERVIEWLibCtl.IExShellFolder,Include As Boolean)
End Sub
```

VBA
```
Private Sub IncludeFolder(ByVal Folder As Object,Include As Boolean)
End Sub
```

VFP
```
LPARAMETERS Folder,Include

```

Xbas…
```
PROCEDURE OnIncludeFolder(oExFolderView,Folder,Include)
RETURN
```

Syntax for IncludeFolder event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="IncludeFolder(Folder,Include)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function IncludeFolder(Folder,Include)
```

```
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComIncludeFolder Variant llFolder Boolean llInclude
    Forward Send OnComIncludeFolder llFolder llInclude
End_Procedure
```

Visual Objects

```
METHOD OCX_IncludeFolder(Folder,Include) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_IncludeFolder(COM _Folder,COMVariant /*bool*/ _Include)
{
}
```

XBasic

```
function IncludeFolder as v (Folder as
OLE::Exontrol.FolderView.1::IExShellFolder,Include as L)
end function
```

dBASE

```
function nativeObject_IncludeFolder(Folder,Include)
return
```

event KeyDown (KeyCode as Integer, Shift as Integer)

Occurs when the user presses a key while an object has the focus.

Type	Description
KeyCode as Integer	An integer that represent the key code.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

Use KeyDown and KeyUp event procedures if you need to respond to both the pressing and releasing of a key. By default, the Refresh method is called when the user presses the F5 key. You test for a condition by first assigning each result to a temporary integer variable and then comparing shift to a bit mask. Use the And operator with the shift argument to test whether the condition is greater than 0, indicating that the modifier was pressed, as in this example:

ShiftDown = (Shift And 1) > 0
CtrlDown = (Shift And 2) > 0
AltDown = (Shift And 4) > 0
In a procedure, you can test for any combination of conditions, as in this example:
If AltDown And CtrlDown Then

Syntax for KeyDown event, **/NET** version, on:

C#
```
private void KeyDown(object sender,ref short KeyCode,short Shift)
{
}
```

VB
```
Private Sub KeyDown(ByVal sender As System.Object,ByRef KeyCode As
Short,ByVal Shift As Short) Handles KeyDown
End Sub
```

Syntax for KeyDown event, **/COM** version, on:

C#
```
private void KeyDownEvent(object sender,
```

```
AxEXFOLDERVIEWLib._IExFolderViewEvents_KeyDownEvent e)
{
}
```

C++

```
void OnKeyDown(short FAR* KeyCode,short Shift)
{
}
```

C++ Builder

```
void __fastcall KeyDown(TObject *Sender,short * KeyCode,short Shift)
{
}
```

Delphi

```
procedure KeyDown(ASender: TObject; var KeyCode : Smallint;Shift : Smallint);
begin
end;
```

Delphi 8 (.NET only)

```
procedure KeyDownEvent(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_KeyDownEvent);
begin
end;
```

Powe…

```
begin event KeyDown(integer KeyCode,integer Shift)
end event KeyDown
```

VB.NET

```
Private Sub KeyDownEvent(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_KeyDownEvent) Handles
KeyDownEvent
End Sub
```

VB6

```
Private Sub KeyDown(KeyCode As Integer,Shift As Integer)
End Sub
```

VBA

```
Private Sub KeyDown(KeyCode As Integer,ByVal Shift As Integer)
End Sub
```

VFP

```
LPARAMETERS KeyCode,Shift
```

Xbas…

```
PROCEDURE OnKeyDown(oExFolderView,KeyCode,Shift)
RETURN
```

Syntax for KeyDown event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="KeyDown(KeyCode,Shift)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function KeyDown(KeyCode,Shift)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComKeyDown Short llKeyCode Short llShift
    Forward Send OnComKeyDown llKeyCode llShift
End_Procedure
```

Visual Objects

```
METHOD OCX_KeyDown(KeyCode,Shift) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_KeyDown(COMVariant /*short*/ _KeyCode,int _Shift)
{
}
```

XBasic

```
function KeyDown as v (KeyCode as N,Shift as N)
end function
```

dBASE

```
function nativeObject_KeyDown(KeyCode,Shift)
return
```

The following VB sample starts renaming the selected folder, when the user presses the F2 key:

```
Private Sub ExFolderView1_KeyDown(KeyCode As Integer, Shift As Integer)
    If (KeyCode = vbKeyF2) Then
        ExFolderView1.SelectedFolder.InvokeRename
```

```
    End If
End Sub
```

The following VB sample displays the object's Properties dialog, when the user presses the F2 key:

```
Private Sub ExFolderView1_KeyDown(KeyCode As Integer, Shift As Integer)
    If (KeyCode = vbKeyF2) Then
        ExFolderView1.SelectedFolder.InvokeCommand ("Properties")
    End If
End Sub
```

event KeyPress (KeyAscii as Integer)

Occurs when the user presses and releases an ANSI key.

Type	Description
KeyAscii as Integer	An integer that returns a standard numeric ANSI keycode.

The KeyPress event lets you immediately test keystrokes for validity or for formatting characters as they are typed. Changing the value of the keyascii argument changes the character displayed. Use KeyDown and KeyUp event procedures to handle any keystroke not recognized by KeyPress, such as function keys, editing keys, navigation keys, and any combinations of these with keyboard modifiers. Unlike the KeyDown and KeyUp events, KeyPress does not indicate the physical state of the keyboard; instead, it passes a character. KeyPress interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters

Syntax for KeyPress event, **/NET** version, on:

C#

```
private void KeyPress(object sender,ref short KeyAscii)
{
}
```

VB

```
Private Sub KeyPress(ByVal sender As System.Object,ByRef KeyAscii As Short)
Handles KeyPress
End Sub
```

Syntax for KeyPress event, **/COM** version, on:

C#

```
private void KeyPressEvent(object sender,
AxEXFOLDERVIEWLib._IExFolderViewEvents_KeyPressEvent e)
{
}
```

C++

```
void OnKeyPress(short FAR* KeyAscii)
{
}
```

C++ Builder

```
void __fastcall KeyPress(TObject *Sender,short * KeyAscii)
{
}
```

Delphi
```
procedure KeyPress(ASender: TObject; var KeyAscii : Smallint);
begin
end;
```

Delphi 8 (.NET only)
```
procedure KeyPressEvent(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_KeyPressEvent);
begin
end;
```

Powe…
```
begin event KeyPress(integer KeyAscii)
end event KeyPress
```

VB.NET
```
Private Sub KeyPressEvent(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_KeyPressEvent) Handles
KeyPressEvent
End Sub
```

VB6
```
Private Sub KeyPress(KeyAscii As Integer)
End Sub
```

VBA
```
Private Sub KeyPress(KeyAscii As Integer)
End Sub
```

VFP
```
LPARAMETERS KeyAscii

```

Xbas…
```
PROCEDURE OnKeyPress(oExFolderView,KeyAscii)
RETURN
```

Syntax for KeyPress event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="KeyPress(KeyAscii)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function KeyPress(KeyAscii)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComKeyPress Short llKeyAscii
    Forward Send OnComKeyPress llKeyAscii
End_Procedure
```

Visual Objects

```
METHOD OCX_KeyPress(KeyAscii) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_KeyPress(COMVariant /*short*/ _KeyAscii)
{
}
```

XBasic

```
function KeyPress as v (KeyAscii as N)
end function
```

dBASE

```
function nativeObject_KeyPress(KeyAscii)
return
```

event KeyUp (KeyCode as Integer, Shift as Integer)

Occurs when the user releases a key while an object has the focus.

Type	Description
KeyCode as Integer	An integer that represent the key code.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of shift is 6.

Use the KeyUp event procedure to respond to the releasing of a key.

Syntax for KeyUp event, **/NET** version, on:

C#

```
private void KeyUp(object sender,ref short KeyCode,short Shift)
{
}
```

VB

```
Private Sub KeyUp(ByVal sender As System.Object,ByRef KeyCode As Short,ByVal
Shift As Short) Handles KeyUp
End Sub
```

Syntax for KeyUp event, **/COM** version, on:

C#

```
private void KeyUpEvent(object sender,
AxEXFOLDERVIEWLib._IExFolderViewEvents_KeyUpEvent e)
{
}
```

C++

```
void OnKeyUp(short FAR* KeyCode,short Shift)
{
}
```

C++ Builder

```
void __fastcall KeyUp(TObject *Sender,short * KeyCode,short Shift)
{
```

```
}
```

Delphi

```
procedure KeyUp(ASender: TObject; var KeyCode : Smallint;Shift : Smallint);
begin
end;
```

Delphi 8 (.NET only)

```
procedure KeyUpEvent(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_KeyUpEvent);
begin
end;
```

Powe…

```
begin event KeyUp(integer KeyCode,integer Shift)
end event KeyUp
```

VB.NET

```
Private Sub KeyUpEvent(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_KeyUpEvent) Handles KeyUpEvent
End Sub
```

VB6

```
Private Sub KeyUp(KeyCode As Integer,Shift As Integer)
End Sub
```

VBA

```
Private Sub KeyUp(KeyCode As Integer,ByVal Shift As Integer)
End Sub
```

VFP

```
LPARAMETERS KeyCode,Shift

```

Xbas…

```
PROCEDURE OnKeyUp(oExFolderView,KeyCode,Shift)
RETURN
```

Syntax for KeyUp event, **/COM** version (others), on:

Java…

```
<SCRIPT EVENT="KeyUp(KeyCode,Shift)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function KeyUp(KeyCode,Shift)
End Function
```

```
</SCRIPT>
```

Visual Data…

```
Procedure OnComKeyUp Short llKeyCode Short llShift
    Forward Send OnComKeyUp llKeyCode llShift
End_Procedure
```

Visual Objects

```
METHOD OCX_KeyUp(KeyCode,Shift) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_KeyUp(COMVariant /*short*/ _KeyCode,int _Shift)
{
}
```

XBasic

```
function KeyUp as v (KeyCode as N,Shift as N)
end function
```

dBASE

```
function nativeObject_KeyUp(KeyCode,Shift)
return
```

event MouseDown (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Fired when the user release one of the buttons mouse.

Type	Description
Button as Integer	An integer that identifies the button that was pressed to cause the event
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The X value is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The Y value is always expressed in container coordinates.

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. Use the FolderFromPoint property to retrieve the folder from the cursor.

Syntax for MouseDown event, **/NET** version, on:

C#

```
private void MouseDownEvent(object sender,short Button,short Shift,int X,int Y)
{
}
```

VB

```
Private Sub MouseDownEvent(ByVal sender As System.Object,ByVal Button As
Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles
MouseDownEvent
End Sub
```

Syntax for MouseDown event, **/COM** version, on:

C#

```
private void MouseDownEvent(object sender,
AxEXFOLDERVIEWLib._IExFolderViewEvents_MouseDownEvent e)
{
```

```
}
```

C++

```
void OnMouseDown(short Button,short Shift,long X,long Y)
{
}
```

C++ Builder

```
void __fastcall MouseDown(TObject *Sender,short Button,short Shift,int X,int Y)
{
}
```

Delphi

```
procedure MouseDown(ASender: TObject; Button : Smallint;Shift : Smallint;X :
Integer;Y : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure MouseDownEvent(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_MouseDownEvent);
begin
end;
```

Powe…

```
begin event MouseDown(integer Button,integer Shift,long X,long Y)
end event MouseDown
```

VB.NET

```
Private Sub MouseDownEvent(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_MouseDownEvent) Handles
MouseDownEvent
End Sub
```

VB6

```
Private Sub MouseDown(ByVal Button As Integer,ByVal Shift As Integer,ByVal X As
Single,ByVal Y As Single)
End Sub
```

VBA

```
Private Sub MouseDown(ByVal Button As Integer,ByVal Shift As Integer,ByVal X As
Long,ByVal Y As Long)
End Sub
```

VFP

```
LPARAMETERS Button,Shift,X,Y
```

Xbas…
```
PROCEDURE OnMouseDown(oExFolderView,Button,Shift,X,Y)
RETURN
```

Syntax for MouseDown event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="MouseDown(Button,Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function MouseDown(Button,Shift,X,Y)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComMouseDown Short llButton Short llShift OLE_XPOS_PIXELS llX
OLE_YPOS_PIXELS llY
    Forward Send OnComMouseDown llButton llShift llX llY
End_Procedure
```

Visual Objects
```
METHOD OCX_MouseDown(Button,Shift,X,Y) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_MouseDown(int _Button,int _Shift,int _X,int _Y)
{
}
```

XBasic
```
function MouseDown as v (Button as N,Shift as N,X as
OLE::Exontrol.FolderView.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.FolderView.1::OLE_YPOS_PIXELS)
end function
```

dBASE
```
function nativeObject_MouseDown(Button,Shift,X,Y)
return
```

Below example displays the folder from the cursor:

```
Private Sub ExFolderView1_MouseMove(ByVal Button As Integer, ByVal Shift As Integer,
ByVal X As Single, ByVal Y As Single)
```

```
    With ExFolderView1
        Dim f As EXFOLDERVIEWLibCtl.ExShellFolder
        Set f = .FolderFromPoint(-1, -1)
        If Not (f Is Nothing) Then
            Debug.Print f.DisplayName
        End If
    End With
End Sub
```

event MouseMove (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Fired when the user move the mouse over the ExFolderView control.

Type	Description
Button as Integer	An integer that corresponds to the state of the mouse buttons in which a bit is set if the button is down
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates.

The MouseMove event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseMove event whenever the mouse position is within its borders. Use the FolderFromPoint property to retrieve the folder from the cursor.

Syntax for MouseMove event, **/NET** version, on:

C#
```
private void MouseMoveEvent(object sender,short Button,short Shift,int X,int Y)
{
}
```

VB
```
Private Sub MouseMoveEvent(ByVal sender As System.Object,ByVal Button As Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles MouseMoveEvent
End Sub
```

Syntax for MouseMove event, **/COM** version, on:

C#
```
private void MouseMoveEvent(object sender,
AxEXFOLDERVIEWLib._IExFolderViewEvents_MouseMoveEvent e)
{
}
```

C++

```
void OnMouseMove(short Button,short Shift,long X,long Y)
{
}
```

C++ Builder

```
void __fastcall MouseMove(TObject *Sender,short Button,short Shift,int X,int Y)
{
}
```

Delphi

```
procedure MouseMove(ASender: TObject; Button : Smallint;Shift : Smallint;X :
Integer;Y : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure MouseMoveEvent(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_MouseMoveEvent);
begin
end;
```

Powe…

```
begin event MouseMove(integer Button,integer Shift,long X,long Y)
end event MouseMove
```

VB.NET

```
Private Sub MouseMoveEvent(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_MouseMoveEvent) Handles
MouseMoveEvent
End Sub
```

VB6

```
Private Sub MouseMove(ByVal Button As Integer,ByVal Shift As Integer,ByVal X As
Single,ByVal Y As Single)
End Sub
```

VBA

```
Private Sub MouseMove(ByVal Button As Integer,ByVal Shift As Integer,ByVal X As
Long,ByVal Y As Long)
End Sub
```

VFP

```
LPARAMETERS Button,Shift,X,Y
```

Xbas…
```
PROCEDURE OnMouseMove(oExFolderView,Button,Shift,X,Y)
RETURN
```

Syntax for MouseMove event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="MouseMove(Button,Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function MouseMove(Button,Shift,X,Y)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComMouseMove Short llButton Short llShift OLE_XPOS_PIXELS llX
OLE_YPOS_PIXELS llY
    Forward Send OnComMouseMove llButton llShift llX llY
End_Procedure
```

Visual Objects
```
METHOD OCX_MouseMove(Button,Shift,X,Y) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_MouseMove(int _Button,int _Shift,int _X,int _Y)
{
}
```

XBasic
```
function MouseMove as v (Button as N,Shift as N,X as
OLE::Exontrol.FolderView.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.FolderView.1::OLE_YPOS_PIXELS)
end function
```

dBASE
```
function nativeObject_MouseMove(Button,Shift,X,Y)
return
```

Below example displays the folder from the cursor:

```
Private Sub ExFolderView1_MouseMove(ByVal Button As Integer, ByVal Shift As Integer,
ByVal X As Single, ByVal Y As Single)
```

```
    With ExFolderView1
        Dim f As EXFOLDERVIEWLibCtl.ExShellFolder
        Set f = .FolderFromPoint(-1, -1)
        If Not (f Is Nothing) Then
            Debug.Print f.DisplayName
        End If
    End With
End Sub
```

event MouseUp (Button as Integer, Shift as Integer, X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS)

Fired when user release the mouse over the ExFolderView control.

Type	Description
Button as Integer	An integer that identifies the button that was pressed to cause the event.
Shift as Integer	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or released.
X as OLE_XPOS_PIXELS	A single that specifies the current X location of the mouse pointer. The x values is always expressed in container coordinates.
Y as OLE_YPOS_PIXELS	A single that specifies the current Y location of the mouse pointer. The y values is always expressed in container coordinates.

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. Use the FolderFromPoint property to retrieve the folder from the cursor.

Syntax for MouseUp event, **/NET** version, on:

C#
```
private void MouseUpEvent(object sender,short Button,short Shift,int X,int Y)
{
}
```

VB
```
Private Sub MouseUpEvent(ByVal sender As System.Object,ByVal Button As Short,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles MouseUpEvent
End Sub
```

Syntax for MouseUp event, **/COM** version, on:

C#
```
private void MouseUpEvent(object sender,
AxEXFOLDERVIEWLib._IExFolderViewEvents_MouseUpEvent e)
{
```

```
}
```

C++

```
void OnMouseUp(short Button,short Shift,long X,long Y)
{
}
```

C++ Builder

```
void __fastcall MouseUp(TObject *Sender,short Button,short Shift,int X,int Y)
{
}
```

Delphi

```
procedure MouseUp(ASender: TObject; Button : Smallint;Shift : Smallint;X :
Integer;Y : Integer);
begin
end;
```

Delphi 8 (.NET only)

```
procedure MouseUpEvent(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_MouseUpEvent);
begin
end;
```

Powe…

```
begin event MouseUp(integer Button,integer Shift,long X,long Y)
end event MouseUp
```

VB.NET

```
Private Sub MouseUpEvent(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_MouseUpEvent) Handles
MouseUpEvent
End Sub
```

VB6

```
Private Sub MouseUp(ByVal Button As Integer,ByVal Shift As Integer,ByVal X As
Single,ByVal Y As Single)
End Sub
```

VBA

```
Private Sub MouseUp(ByVal Button As Integer,ByVal Shift As Integer,ByVal X As
Long,ByVal Y As Long)
End Sub
```

VFP

```
LPARAMETERS Button,Shift,X,Y
```

Xbas…
```
PROCEDURE OnMouseUp(oExFolderView,Button,Shift,X,Y)
RETURN
```

Syntax for MouseUp event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="MouseUp(Button,Shift,X,Y)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function MouseUp(Button,Shift,X,Y)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComMouseUp Short llButton Short llShift OLE_XPOS_PIXELS llX
OLE_YPOS_PIXELS llY
    Forward Send OnComMouseUp llButton llShift llX llY
End_Procedure
```

Visual Objects
```
METHOD OCX_MouseUp(Button,Shift,X,Y) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_MouseUp(int _Button,int _Shift,int _X,int _Y)
{
}
```

XBasic
```
function MouseUp as v (Button as N,Shift as N,X as
OLE::Exontrol.FolderView.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.FolderView.1::OLE_YPOS_PIXELS)
end function
```

dBASE
```
function nativeObject_MouseUp(Button,Shift,X,Y)
return
```

Below example displays the folder from the cursor:

```
Private Sub ExFolderView1_MouseMove(ByVal Button As Integer, ByVal Shift As Integer,
ByVal X As Single, ByVal Y As Single)
```

```
    With ExFolderView1
        Dim f As EXFOLDERVIEWLibCtl.ExShellFolder
        Set f = .FolderFromPoint(-1, -1)
        If Not (f Is Nothing) Then
            Debug.Print f.DisplayName
        End If
    End With
End Sub
```

event QueryContextMenu (Folder as ExShellFolder, Items as String, Separator as String)

Fired when the context menu is about to be active.

Type	Description
Folder as ExShellFolder	A Folder object whose context menu is displayed
Items as String	A string expression that specifies the context menu items to display.
Separator as String	A string expression that contains the separator text to use in the Items parameter.

Fires when the context menu is about to display. This allows you to customize the control context menu. You can then react to the context menu through the BeforeShellMenuCommand and AfterShellMenuCommand menus.

Syntax for QueryContextMenu event, **/NET** version, on:

C#
```
private void QueryContextMenu(object sender,exontrol.EXFOLDERVIEWLib.ExShellFolder Folder,ref string Items,ref string Separator)
{
}
```

VB
```
Private Sub QueryContextMenu(ByVal sender As System.Object,ByVal Folder As exontrol.EXFOLDERVIEWLib.ExShellFolder,ByRef Items As String,ByRef Separator As String) Handles QueryContextMenu
End Sub
```

Syntax for QueryContextMenu event, **/COM** version, on:

C#
```
private void QueryContextMenu(object sender, AxEXFOLDERVIEWLib._IExFolderViewEvents_QueryContextMenuEvent e)
{
}
```

C++
```
void OnQueryContextMenu(LPDISPATCH Folder,LPCTSTR FAR* Items,LPCTSTR FAR* Separator)
{
}
```

C++ Builder

```
void __fastcall QueryContextMenu(TObject
*Sender,Exfolderviewlib_tlb::IExShellFolder *Folder,BSTR * Items,BSTR * Separator)
{
}
```

Delphi

```
procedure QueryContextMenu(ASender: TObject; Folder : IExShellFolder;var Items
: WideString;var Separator : WideString);
begin
end;
```

Delphi 8 (.NET only)

```
procedure QueryContextMenu(sender: System.Object; e:
AxEXFOLDERVIEWLib._IExFolderViewEvents_QueryContextMenuEvent);
begin
end;
```

Powe…

```
begin event QueryContextMenu(oleobject Folder,string Items,string Separator)
end event QueryContextMenu
```

VB.NET

```
Private Sub QueryContextMenu(ByVal sender As System.Object, ByVal e As
AxEXFOLDERVIEWLib._IExFolderViewEvents_QueryContextMenuEvent) Handles
QueryContextMenu
End Sub
```

VB6

```
Private Sub QueryContextMenu(ByVal Folder As
EXFOLDERVIEWLibCtl.IExShellFolder,Items As String,Separator As String)
End Sub
```

VBA

```
Private Sub QueryContextMenu(ByVal Folder As Object,Items As String,Separator
As String)
End Sub
```

VFP

```
LPARAMETERS Folder,Items,Separator

```

Xbas…

```
PROCEDURE OnQueryContextMenu(oExFolderView,Folder,Items,Separator)
RETURN
```

Syntax for QueryContextMenu event, **/COM** version (others), on:

Java…
```
<SCRIPT EVENT="QueryContextMenu(Folder,Items,Separator)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function QueryContextMenu(Folder,Items,Separator)
End Function
</SCRIPT>
```

Visual Data…
```
Procedure OnComQueryContextMenu Variant llFolder String llItems String
llSeparator
    Forward Send OnComQueryContextMenu llFolder llItems llSeparator
End_Procedure
```

Visual Objects
```
METHOD OCX_QueryContextMenu(Folder,Items,Separator) CLASS MainDialog
RETURN NIL
```

X++
```
void onEvent_QueryContextMenu(COM _Folder,COMVariant /*string*/
_Items,COMVariant /*string*/ _Separator)
{
}
```

XBasic
```
function QueryContextMenu as v (Folder as
OLE::Exontrol.FolderView.1::IExShellFolder,Items as C,Separator as C)
end function
```

dBASE
```
function nativeObject_QueryContextMenu(Folder,Items,Separator)
return
```

The following sample code adds two new menu items to the folder context menu.

```
Private Sub ExFolderView1_QueryContextMenu(ByVal Folder As
EXFOLDERVIEWLibCtl.IExShellFolder, Items As String, Separator As String)
    Items = Separator & "New Item1" & Separator & "New Item2"
End Sub
```