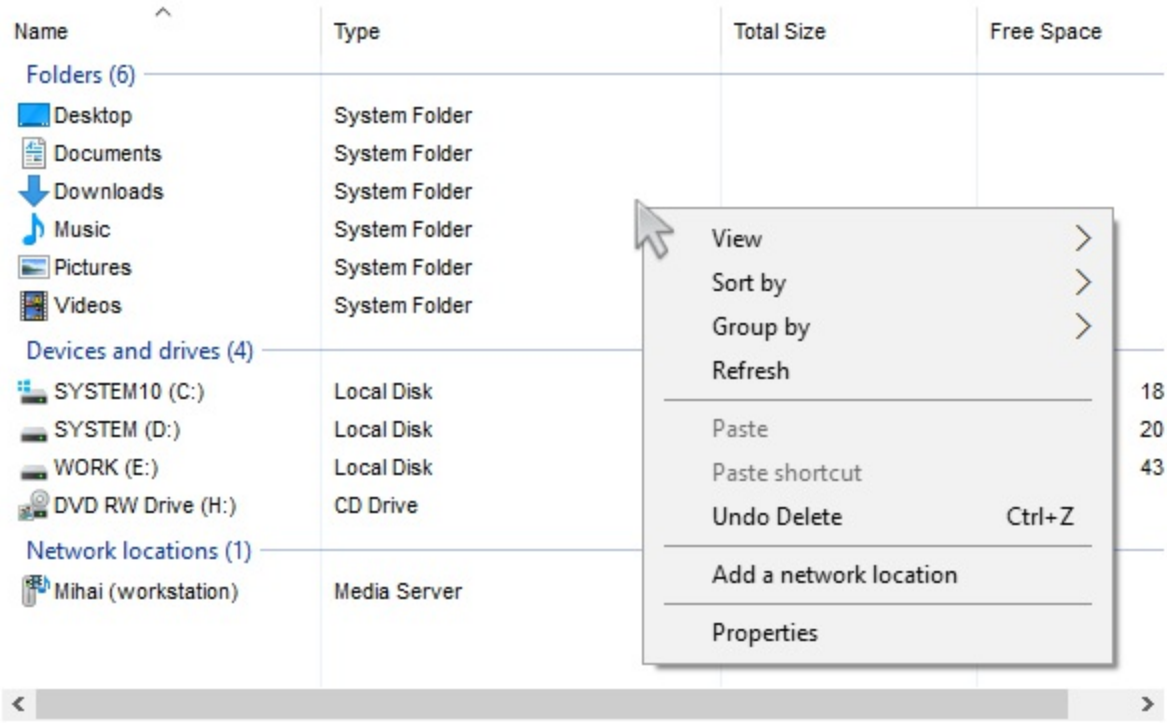
ExShellView

Exontrol's new ExShellView component provides a file list view which is identical with the right pane of your Windows Explorer. The ExShellView can be used in conjunction with Exontrol's ExFolderView to create applications which have complete - or limited - explorer capabilities. Using ExShellView you can easily present a list of files to users which they can, optionally, be allowed to display as large icons, small icons, names, or complete details. There is a FilePattern property you can use to select which files will be displayed. And, for more demanding filtering requirements an event is fired for each item before it is displayed, enabling you to determine on the fly which items to display. There is a pattern matching method to help you filter names, and many other useful properties and events.

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 AllowContextMenuEnum

The AllowContextMenuEnum type defines the value the AllowContextMenu property supports. The AllowContextMenu property enables or disables the control's context-menu. The AllowContextMenuEnum type supports the following values:

Name	Value	Description
exDisableContextMenu	0	Disable the control's context-menu.
exAllowContextMenu	-1	Allow the control's context-menu.
exAllowListViewContextMenu	1	Allow the list-view's context-menu.
exAllowHeaderContextMenu	2	Allow the header's context-menu.

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

Indicates the attributes of the selected object. The Attribute property indicates the object's attributes.

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

Masks a collection of attributes. Use the Attributes property to retrieve specified attributes based on 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 FolderFlagsEnum

The FolderFlagsEnum type specifies the control's behavior. Use the ModifyFolderFlags method to add or remove flags to the current view.

Name	Value	Description
NoFlag	0	None
AutoArrange	1	Automatically arrange the elements in the view
DesktopStyle	32	Make the folder behave like the desktop
SingleSel	64	Do not allow more than a single item to be selected
NoSubFolders	128	Do not show subfolders
Transparent	256	Draw transparently, This is used only for the desktop
NoClientEdge	512	Do not add the WS_EX_CLIENTEDGE value to the view
NoScroll	1024	Do not add scroll bars. This is used only for the desktop
AlignLeft	2048	The view should be left-aligned
NoIcons	4096	The view should not display icons

constants IncludeObjectEnum

The IncludeObjectEnum type specifies the list of objects being included in the control's list. Use the IncludeObjectType property to specify the object being included in the control's list.

Name	Value	Description
AllObjects	0	Don't send the IncludeObject event.
UserObjects	1	Send the IncludeObject event, and let the user to choose.
FoldersOnly	2	Select only folders, and the send the IncludeObject event to let the user to choose.
PatternObjects	3	Select all folders, and choose only files that macthing with a given pattern.

constants ObjectTypeEnum

The ObjectTpeEnum type indicates the list of objects being requested. Use the Get method to retrieve the collection of selected item as well as all items displayed in the current view.

Name	Value	Description
NoItems	0	Nothing to collect.
SelectedItems	1	Collects the list of files/folders being selected.
AllItems	2	Gets the entire collection of files/folders.
AsDisplayed	16	AsDisplayed flag can be combined with SelectedItems or AllItems to get the selected/all items as they are listed

constants SelectItemFlagsEnum

The SelectItemFlagsEnum type specifies the actions that SelectItem method may execute.

Name	Value	Description
Deselect	0	Deselect the specified item
EnsureVisible	8	Ensure the item is displayed on the screen.
Focused	16	The item should be given the focus.
Select	1	The item should be selected

constants SpecialFolderPathConstants

Indicates the list of common folders in your system. Use the SpecialFolder property to build an ExShellFolder object based on a common folder.

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	18	Network Neighborhood virtual folder representing 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

constants StatesEnum

The StatesEnum type indicates the view's state that's being changed. The StateChange event notifies your application whether the view is changing its state.

Name	Value	Description
SetFocusState	0	The focus has been set to the view
KillFocusState	1	The view has lost the focus
SelChangeState	2	The selection has changed
RenameState	3	An item has been renamed.

constants ViewModeType

The ViewModeType expression indicates how the control displays its items. Use the ViewMode property to change the control's view mode.

Name	Value	Description
LargeIcons	1	The view should display large icons
SmallIcon	2	The view should display small icons
List	3	Object names are displayed in a list view
Details	4	Object names and other selected information, such as the size or date last updated, are shown
Thumbnail	5	Shows the view as thumbnail
Tile	6	Shows the view as tiles
Thumbstrip	7	Shows the view as thumb strip
Content	8	Shows the view as content
Extra_Large_Icons	13	Shows the view as extra large icons (Available for Windows Vista, Windows 7, ...)
Large_Icons	14	Shows the view as large icons (Available for Windows Vista, Windows 7, ...)
Medium_Icons	15	Shows the view as medium icons (Available for Windows Vista, Windows 7, ...)

ExShellFolder object

Contains information about a folder. The ExShellFolder object holds information about a shell folder. This information includes the name, the path, and the ID of the contained shell folder. The BrowseFolder property retrieves a ExShellFolder object that contains information about browsed shell folder. The ID property returns a unique identifier generated by Windows that represents this Folder object. This identifier is used by Windows to represent any shell object. The Name property holds the name of the folder and the path represents the path of the folder.

Name	Description
ID	Retrieves the Folder's ITEMIDLIST
Name	Retrieves the name of the object.
Path	Retrieves the full path of the source.

property ExShellFolder.ID as Variant

Retrieves the Folder's ITEMIDLIST

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

The ID property retrieves a string that contains the unique shell object identifier. The identifier is generated by Windows and is used to represent any shell object.

property ExShellFolder.Name as String

Retrieves the name of the object.

Type	Description
String	Indicates the folder's name.

This property holds the name of the folder, that is the same as folder's name on a disk.

property ExShellFolder.Path as String

Retrieves the full path of the source.

Type	Description
String	A String expression that indicates the full path of the folder.

This property holds the path of the folder. Since each ExShellFolder object is usually a file on a disk, this property determines where it is stored.

ExShellObject object

Holds information about an individual shell object. The ExShellObject is an element of the ExShellObjects collection. The ExShellObject holds information about a shell object. It contains a name, a path and atributes of the contained shell object. The ExShellObject object can be retrieved using the Item property of ExShellObjects collection. For instance, when the you want to get the collection of selected items, you need to get a reference to ExShellObjects collection by using the Objects property of ExShellView control. Once that you have this reference, you have to ask the collection for selected items by calling the Get(SelectedItems) method of ExShellObjects collection. Now, the ExShellObjects contains a collection files and folders selected. To determine the type of a shell object, you use the Attribute property.

Name	Description
Attribute	Check if anAttribute is set for this object
Attributes	Retrieve the attributes of this object
InvokeCommand	Invokes a specified command from the object's context menu.
InvokeRename	Renames a shell object at runtime.
Name	Return the name of object, relative to parent folder
Path	Retrieve the full path of one object
SelectItem	Changes the selection state of one item within the shell view window.

property ExShellObject.Attribute (Attribute as AttributesEnum) as Boolean

property Attribute - Check if anAttribute is set for this object

Type	Description
Attribute as AttributesEnum	A constant value that is used to determine an object's attribute.
Boolean	A Boolean expression that indicates whether selected object contains specified attribute.

Use this property to determine if the current object is, a Folder, a short-cut, is visible, can be deleted, and so on.

property ExShellObject.Attributes (Mask as AttributesMask) as Long

property Attributes - Retrieve the attributes of this object

Type	Description
Mask as AttributesMask	A constant value that is used to determine the object's combination of attributes
Long	A long expression that indicates the attributes being requested.

This property returns one or more of an object's attributes. The return value will be a combination of all the object's attributes.

method ExShellObject.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 object's context menu. Use the InvokeRename method to rename an object at runtime. Use the Get(SelectedItems) method to retrieve the selected objects in the current view.

Here's the list of the identifiers for some known items in the object's context menu :

- Create Shortcut **(17)**
- Delete **(18)**
- Properties **(20)**
- Cut **(25)**
- Copy **(26)**

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

```
Private Sub ExShellView1_KeyDown(KeyCode As Integer, Shift As Integer)
  If KeyCode = vbKeyF2 Then
    ExShellView1.Objects.Get (SelectedItems)
    With ExShellView1.Objects
      If (.Count > 0) Then
        .Item(0).InvokeCommand ("Properties")
      End If
    End With
  End If
End Sub
```

method ExShellObject.InvokeRename ()

Renames a shell object at runtime.

Type	Description

Call the InvokeRename method to call renaming an folder or a file, at run-time. Use the Get(SelectedItems) method to retrieve the selected objects in the current view. The rename operation starts only if the selected shell object supports renaming. For instance, if you try to rename the Recycle Bin folder, it is not allowed, since it doesn't support renaming. The view is focused, when the InvokeRename method is executed. Use the InvokeCommand method to execute a command from the object's context menu.

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

```
Private Sub ExShellView1_KeyDown(KeyCode As Integer, Shift As Integer)
  If KeyCode = vbKeyF2 Then
    ExShellView1.Objects.Get (SelectedItems)
    With ExShellView1.Objects
      If (.Count > 0) Then
        .Item(0).InvokeRename
      End If
    End With
  End If
End Sub
```

property ExShellObject.Name as String

Return the name of object, relative to parent folder

Type	Description
String	A String expression that specified the object's name.

Since every item needs to have it's name, this property holds it. This value is the same text as seen in eXShellView when browsed.

property ExShellObject.Path as String

Retrieve the full path of one object

Type	Description
String	A string expression that indicates the path to the object.

Indicates the path where the object is stored. This property returns complete (absolute) path of referred item. If only short (relative) path is needed, Name property should be used.

method ExShellObject.SelectItem (Flags as SelectItemFlagsEnum)

Changes the selection state of one item within the shell view window.

Type	Description
Flags as SelectItemFlagsEnum	A SelectItemFlagsEnum expression that specifies the action being executed.

Use the SelectItem property to ensure that a specified item is visible.

ExShellObjects object

Holds a collection of ExShellObject objects. Use the Objects property to get the entire list of browsed items, or selected items, and so on.

Name	Description
Count	Counts the element from collection.
Get	Refreshes the Objects collection.
Item	Retrieves the ExShellObject object given its index.

property ExShellObjects.Count as Long

Counts the element from collection.

Type	Description
Long	A long expression that specifies the count of elements in the collection.

This read-only property returns the number if elements in the ExShellObjects collection. Use the Item property to access an element in the collection. Use the Get method before accession the Count or Item property to retrieve the specific files or folders.

method ExShellObjects.Get (objectType as ObjectTypeEnum)

Refreshes the Objects collection.

Type	Description
objectType as ObjectTypeEnum	An ObjectTypeEnum expression that specifies the list of items being requested.

The Get method fills the ExShellObjects collection with specific files/folders (selection or all items). The Get method can clear the collection, fill it with selected items, or fill it with all files/folders in the control. The Count property indicates the number of elements within the collection. The Item property retrieves a specific member in the collection.

The Get method gets:

- nothing, if the objectType parameter is **NoItems**
- all files or folders being listed in the current view, if the objectType parameter is **AllItems**
- all files or folders being listed in the current view, as they are displayed, if the objectType parameter is **AllItems Or AsDisplayed**
- selected files or folders, if the objectType parameter is **SelectedItems**
- selected files or folders as they are displayed, if the objectType parameter is **SelectedItems or AsDisplayed**

The following VB sample gets the files being selected as they are displayed (sorted, ...)

```
With ExShellView1
    .Objects.Get (SelectedItems Or AsDisplayed)
    For i = 0 To .Objects.Count - 1
      Debug.Print .Objects(i).Name
    Next
End With
```

The following VB.NET sample shows how to get the selected files/folder for /NET assembly version:

```
Dim i As Long = 0, s As String = ""
With Exshellview1
  .Objects.Get(exontrol.EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems)
```

```
    With .Objects
        For i = 0 To .Count - 1
            Dim sel As exontrol.EXSHELLVIEWLib.exshellobject = .Item(i)
            ' * The sel indicates the shell object being selected *
            s = s + sel.Name + vbCrLf
        Next
    End With
End With
If s.Length > 0 Then
    MessageBox.Show(s, "Selection")
Else
    MessageBox.Show("Empty", "Selection")
End If
```

The following C# sample shows how to get the selected files/folder for /NET assembly version:

```
string s = "";
exshellview1.Objects.Get(exontrol.EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems);
for ( int i = 0; i < exshellview1.Objects.Count; i++ )
{
    exontrol.EXSHELLVIEWLib.exshellobject sel = exshellview1.Objects[i];
    // * The sel indicates the shell object being selected *
    s = s + sel.Name + "\r\n";
}
if (s.Length > 0)
    MessageBox.Show(s, "Selection");
else
    MessageBox.Show("Empty", "Selection");
```

The following VB.NET sample shows how to get the selected files/folder for /COM on Window.Forms version:

```
Dim i As Long = 0, s As String = ""
With AxExShellView1
    .Objects.Get(EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems)
    With .Objects
        For i = 0 To .Count - 1
```

```
        Dim sel As EXSHELLVIEWLib.ExShellObject = .Item(i)
        ' * The sel indicates the shell object being selected *
        s = s + sel.Name + vbCrLf
      Next
   End With
End With
If s.Length > 0 Then
   MessageBox.Show(s, "Selection")
Else
   MessageBox.Show("Empty", "Selection")
End If
```

The following C# sample shows how to get the selected files/folder for /COM on Window.Forms version:

```
string s = "";
axExShellView1.Objects.Get(EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems);
for (int i = 0; i < axExShellView1.Objects.Count; i++)
{
   EXSHELLVIEWLib.ExShellObject sel = axExShellView1.Objects[i];
   // * The sel indicates the shell object being selected *
   s = s + sel.Name + "\r\n";
}
if (s.Length > 0)
   MessageBox.Show(s, "Selection");
else
   MessageBox.Show("Empty", "Selection");
```

The following VB6 sample shows how to get the selected files/folder for /COM version:

```
Dim i As Long, s As String
s = ""
With ExShellView1
   .Objects.Get (EXSHELLVIEWLibCtl.ObjectTypeEnum.SelectedItems)
   With .Objects
      For i = 0 To .Count - 1
         Dim sel As EXSHELLVIEWLibCtl.ExShellObject
         Set sel = .Item(i)
```

```
        ' * The sel indicates the shell object being selected *
        s = s + sel.Name + vbCrLf
      Next
   End With
End With
If Len(s) > 0 Then
   MsgBox s, , "Selection"
Else
   MsgBox "Empty", , "Selection"
End If
```

The following Access sample shows how to get the selected files/folder for /COM version:

```
Dim i As Long, s As String
s = ""
With ExShellView1
   .Objects.Get (EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems)
   With .Objects
      For i = 0 To .Count - 1
         Dim sel As EXSHELLVIEWLib.ExShellObject
         Set sel = .Item(i)
         ' * The sel indicates the shell object being selected *
         s = s + sel.Name + vbCrLf
      Next
   End With
End With
If Len(s) > 0 Then
   MsgBox s, , "Selection"
Else
   MsgBox "Empty", , "Selection"
End If
```

The following VPF sample shows how to get the selected files/folder for /COM version:

```
local sel
s = ""
with thisform.ExShellView1
   .Objects.Get(1)
```

```
    for i = 0 to .Objects.Count - 1
        sel = .Objects.Item(i)
        s = s + sel.Name + chr(13)+chr(10)
    next
endwith
messagebox(s)
```

The following C++ sample shows how to get the selected files/folder for /COM version:

```
CString s;
CExShellObjects objects = m_shellView.GetObjects();
objects.Get( 1 );
for ( long i = 0; i < objects.GetCount(); i++ )
{
    CExShellObject sel = objects.GetItem( COleVariant( i ) );
    s = s + sel.GetName() + _T("\r\n");

}
if ( s.GetLength() > 0 )
    MessageBox( s, _T("Selection") );
else
    MessageBox( _T("Empty"), _T("Selection") );
```

property ExShellObjects.Item (Index as Variant) as ExShellObject

Retrieves the ExShellObject object given its index.

Type	Description
Index as Variant	A Long expression that specifies the index of the object being requested
ExShellObject	An ExShellObject object being requested.

This objects are created internally and the user cannot add or remove them. The Item property retrieves an object based on its index. Use the Get method to fill the Objects collection with specified files or folders.

The following VB sample displays all browsed objects:

```
ExShellView1.Objects.Get AllItems
With ExShellView1.Objects
   For i = 0 To .Count - 1
      Debug.Print .Item(i).Name
   Next
End With
```

ExShellView 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: {B4E1F234-AF0D-4EAD-8113-A563B40E71CA}. The object's program identifier is: "Exontrol.ShellView". The /COM object module is: "ExShellView.dll"

Exontrol's new ExShellView component provides a file list view which is identical with the right pane of your Windows Explorer. The control supports the following properties and methods:

Name	Description
AlignToGrid	Specifies whether in icon view, icons automatically snap into a grid.
AllowContextMenu	Enables or disables the control's context-menu.
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.
AutoArrange	Specifies whether the files are automatically kept arranged in icon and small icon view.
BrowseFiles	Specifies a list (separated by | character) or safe array of files to be shown in the shell view.
BrowseFolder	Retrieves or sets the browsed folder using a path, a special folder constant or another Folder object.
CancelObjectSelect	Cancels opening the object being double clicked (aka folder or zip files) during the ObjectSelect event.
CanRename	Retrieves or sets a value indicating whether the user can rename files/folders at runtime.
ColumnNames	Defines a list of column's name replacements, separated by comma as 'Name1(NewName1),...'
CurrentFolder	Retrieves or sets the folder to be browsed.
DefaultMenuItems	Retrieves or sets a value that indicates whether the control allows default shell context menu items.
DisableDragDrop	Disables the OLE Drag and Drop within the control.
DrawGridLines	Specifies whether the control shows the grid lines around items, when the control's view is details.
Enabled	Retrieves or sets a value indicating whether the control responds to the user-generated events.

EventParam	Retrieves or sets a value that indicates the current's event parameter.
ExecuteTemplate	Executes a template and returns the result.
FilePattern	Retrieves or sets a string value that indicates the file pattern used to include files. "*.jpg *.bmp "
Font	Retrieves or sets a Font object used to paint the items.
HeaderVisible	Specifies whether the view's header is visible or hidden.
HideFileNames	Specifies whether the files hides the names in icon and small icon view.
HideToolTips	Specifies whether the file displays a tooltip when the cursor hovers the shellview.
hWnd	Retrieves the window handle.
IncludeObjectType	Retrieves or sets the way how the control will filter the objects.
MatchPattern	Checks if the given word matches the given mask.
ModifyFolderFlags	Adds or removes flags that indicates the options for browsed folder.
Objects	Retrieves a collection of ExShellObject objects that indicates the current selection, or all items from the view.
OverlayIcons	Retrieves or sets a value indicating whether the control displays the overlay icons.
Refresh	Refreshes the content of the browsed folder.
SelectAll	Selects or unselects all files in the control when the multiple selection is enabled.
ShellFolder	Retrieves a Folder object based on a path, on a special folder constant or on an ID property.
SpecialFolder	Retrieves a Folder object 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.
UpOneLevel	Browses the parent of current browsed folder.
Version	Retrieves the Version of the control.
ViewFolderFlags	Retrieves the flags for the browsed folder.

ViewMode	Returns or changes the current view of the control.

property ExShellView.AlignToGrid as Boolean

Specifies whether in icon view, icons automatically snap into a grid.

Type	Description
Boolean	A Boolean expression that indicates whether the view aligns the files up to a grid.

By default, the AlignToGrid property is False. Use the AlignToGrid property to specifies whether in icon view, icons automatically snap into a grid. The AutoArrange property specifies whether the files are automatically kept arranged in icon and small icon view.

The AlignToGrid property has effect if the control's ViewMode property is:

- LargeIcons
- SmallIcon
- Thumbnail
- Tile
- Extra_Large_Icons
- Large_Icons
- Medium_Icons

The AlignToGrid property has NO effect if the control's ViewMode property is:

- List
- Details

property ExShellView.AllowContextMenu as AllowContextMenuEnum

Enables or disables the control's context-menu.

Type	Description
AllowContextMenuEnum	An AllowContextMenuEnum expression that specifies flags to apply

By default, the AllowContextMenu property is -1 (exAllowContextMenu), which indicates that the control's context-menu is allowed. The control's context-menu is shown once the user right clicks the control. The AllowContextMenu property on 0 (exDisableContextMenu) disables the control's context-menu, so the user can't show the control's context-menu when user right-clicks the control or press the Context key.

property ExShellView.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.

method ExShellView.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 ExShellView1_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 ExShellView.AutoArrange as Boolean

Specifies whether the files are automatically kept arranged in icon and small icon view.

Type	Description
Boolean	A Boolean expression that indicates whether the files are automatically kept arranged in icon and small icon view.

By default, the AutoArrange property is False. The AutoArrange property specifies whether the files are automatically kept arranged in icon and small icon view. Use the AlignToGrid property to specifies whether in icon view, icons automatically snap into a grid.

The AutoArrange property has effect if the control's ViewMode property is:

- LargeIcons
- SmallIcon
- Thumbnail
- Tile
- Extra_Large_Icons
- Large_Icons
- Medium_Icons

The AutoArrange property has NO effect if the control's ViewMode property is:

- List
- Details

property ExShellView.BrowseFiles as Variant

Specifies a list of files (separated by | character) to be displayed in the shell view.

Type	Description
Variant	The parameter could be one of the following: • A String expression that indicates the list of files to be shown in the shell view. The list of files is separated by | character. You can specify existing files or non-existing files. If the file exists its view is based on its content, while if the file is not-existing, the extension of the file specifies what icon is displayed in the view. Each existing file can display a different icon, if specifying a different extension after the > character. Each file can have a different extension, by adding > extension to the name of the file, such as "C:\exontrol\images\auction.gif>PNG" which will show the auction item with the PNG associated icon. The "C:\Program Files\Microsoft Visual FoxPro 9\vfp9.exe|C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE\devenv.exe" shows the VFP9 and DEVENV tools in the same view • A Safe array of files to be shown on the shell view. Each file can have a different extension, by adding > extension to the name of the file. For instance, the Array("document.doc", "text.txt", "executable.exe") displays a three items

The BrowseFiles property displays files from different folders in the same view. The ViewMode property specifies the current's view mode. The Objects.Get(AllItems) property to get the collection of files/folders being shown in the shell view. The Objects.Get(SelectedItems) property to get a collection of selected files/folders. The ObjectSelected event notifies your application that a new items in the list is selected. The HeaderVisible property specifies whether the shell view shows or hides the control's header. The DrawGridLines property indicates whether the control shows or hides the grid lines around the items. The BrowseFolder property specifies the folder to be shown on the shell view.

The following VB sample shows files giving a safe array:

```
With ExShellView1
```

```
    .ViewMode = Thumbnail
    .BrowseFiles = Array("A.txt", "B.txt", "C.txt")
End With
```

The following VB sample shows the files giving a string:

```
With ExShellView1
    .ViewMode = Thumbnail
    .BrowseFiles = "A.txt|B.txt|C.txt"
End With
```

property ExShellView.BrowseFolder as Variant

Retrieves or sets the browsed folder using a path, a special folder constant or another Folder object.

Type	Description
Variant	A reference to the folder object that is currently browsed.

The BrowseFolder property can be used to specify a different folder to be browsed or can be used to link controls as: ExShellView, ExFolderView, ExFileView and so on. Use the CurrentFolder property to specify the current/browsed folder giving the path as string. BrowseFolder property holds a reference to the ExShellFolder object being browsed, which keeps information about the current folder name, path and so on. The BrowseFolder property is automatically updated as soon as the user browses for a new folder. For instance, double click a folder. The BrowseFolder **does not return** the selected shell objects, instead use the Objects property as shown bellow. Use the ShellFolder or SpecialFolder properties to create ExShellFolder objects based on path, identifier. The ViewMode property specifies the current's view mode. The Objects.Get(SelectedItems) property to get a collection of selected files/folders. The BrowseFolderChange event notifies your application once the user changes the current folder.

Use the BrowseFiles property to display files from different folders in the same view.

For instance, the following VB sample browses the c:\temp folder:

```
With ExShellView1
   .BrowseFolder = .ShellFolder("c:\temp")
End With
```

of the following VB sample browses the Control Panel objects:

```
With ExShellView1
   .BrowseFolder = .SpecialFolder(ControlPanel)
End With
```

The following sample VB sample displays the path of the selected shell object (file or folder) (single selected object):

```
With ExShellView1
   .Objects.Get SelectedItems
   With .Objects
      If .Count > 0 Then
```

```
        Debug.Print .Item(0).Path
      End If
    End With
End With
```

The following sample VB sample displays the path for all selected shell objects (file or folder) (multiple selected objects):

```
Dim i As Long
With ExShellView1
    .Objects.Get SelectedItems
    With .Objects
      For i = 0 To .Count - 1
        Debug.Print .Item(i).Path
      Next
    End With
End With
```

method ExShellView.CancelObjectSelect ()

Cancels opening the object being double clicked (aka folder or zip files) during the ObjectSelect event.

Type	Description

The CancelObjectSelect method has effect only if it is called during the ObjectSelect event. Use the CancelObjectSelect method to prevent opening or browsing for folder items being double clicked. By default, if a folder item is being double clicked, the folder gets browsed. If a file is being double clicked, nothing is happen.

The following VB sample prevents opening a zip file (which is considered a folder item):

```
Private Sub ExShellView1_ObjectSelect(ByVal Object As
EXSHELLVIEWLibCtl.IExShellObject)
   If Object.Name Like "*.zip" Then
      ExShellView1.CancelObjectSelect
   End If
End Sub
```

property ExShellView.CanRename as Boolean

Retrieves or sets a value indicating whether the user can rename files/folders at runtime.

Type	Description
Boolean	A Boolean expression that specifies whether the user can rename the files/folders at runtime.

By default, the CanRename property is False. Use the CanRename property on False, to prevent renaming the files / folders when user browsing a folder. Use the Refresh method to update the view and its settings.

property ExShellView.ColumnNames as String

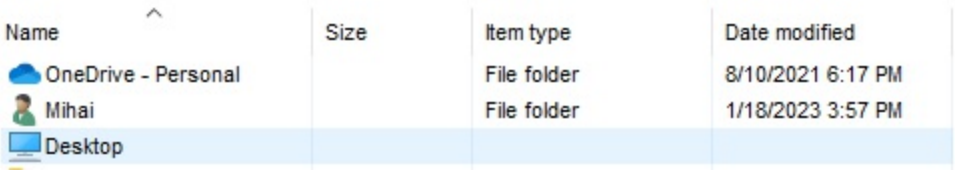
Defines a list of column's name replacements, separated by comma as 'Name1(NewName1),...'

Type	Description
String	A string expression that defines the column-names to rename

By default, the ColumnNames property is "" (empty string). The ColumnNames property renames the column's name. For instance, the "Name(Ime),Date modified(Datum),Item type(Tip),Size(Velikost)" renames the following columns to:

- "Name" to "Ime"
- "Date modified" to "Datum"
- "Item type" to "Tip"
- "Size" to "Velikost"

The following screen show shows the default control:



and with columns renamed:

property ExShellView.CurrentFolder as String

Retrieves or sets the folder to be browsed.

Type	Description
String	A String expression that specifies the current folder.

The CurrentFolder property indicates the path of the folder being browsed. Use the CurrentFolder property to specify the current/browsed folder giving the path as string. Use the BrowseFolder property to specify a browsed folder or to link the ExShellView control with ExFileView or ExFolderView controls. The CurrentFolder property can be seen as a simpler method of BrowseFolder property. The ViewMode property specifies the current's view mode. The Objects.Get(SelectedItems) property to get a collection of selected files/folders. The BrowseFolderChange event notifies your application once the user changes the current folder.

property ExShellView.DefaultMenuItems as Boolean

Retrieves or sets a value that indicates whether the control allows default shell context menu items.

Type	Description
Boolean	A Boolean expression that indicates whether the control displays the shell context menu.

By default, the DefaultMenuItems property is True. Use the DefaultMenuItems property to disable showing the control's shell context menu when user right clicks the list. Use the QueryContextMenu event to add new items to the default shell context menu. The InvokeItemMenu event notifies the application once the user selects a command by identifier in the context menu.

property ExShellView.DisableDragDrop as Boolean

Disables the OLE Drag and Drop within the control.

Type	Description
Boolean	A Boolean expression that specifies whether the OLE Drag and Drop within the control is enabled or disabled.

By default, the DisableDragDrop property is False. You can use the DisableDragDrop property to disable OLE Drag and Drop within the control.

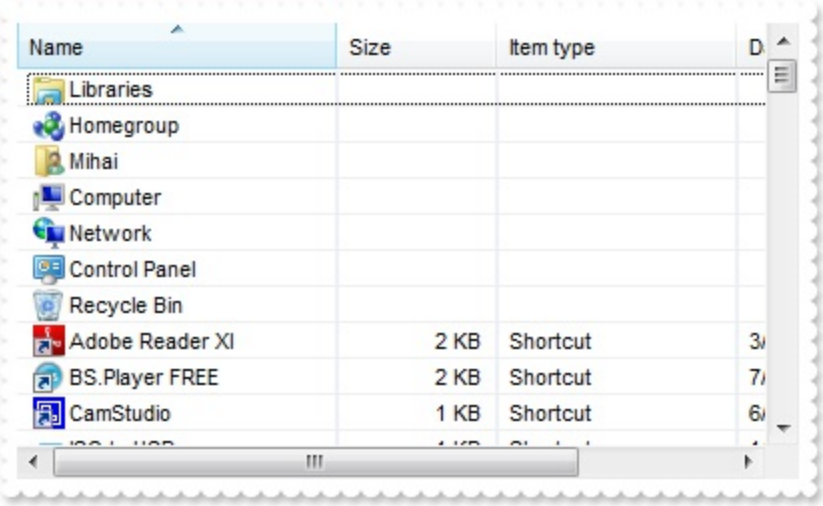
property ExShellView.DrawGridLines as Boolean

Specifies whether the control shows the grid lines arround items, when the control's view is details.

Type	Description
Boolean	A Boolean expression that specifies whether the control shows the grid lines around items/files/folders.

By default, the DrawGridLines property is False, which indicates that no grid lines are displayed around the items. The DrawGridLines property has effect while the control's ViewMode property is Details.

The following screen shot shows the view with grid lines around the files/folders:



The following samples shows the grid lines around the files/folders:

VBA (MS Access, Excell...)

```
With ExShellView1
   .ViewMode = 4
   .DrawGridLines = True
   .Refresh
End With
```

VB6

```
With ExShellView1
   .ViewMode = Details
   .DrawGridLines = True
   .Refresh
```

```
End With
```

VB.NET

```
With Exshellview1
    .ViewMode = exontrol.EXSHELLVIEWLib.ViewModeType.Details
    .DrawGridLines = True
    .Refresh()
End With
```

VB.NET for /COM

```
With AxExShellView1
    .ViewMode = EXSHELLVIEWLib.ViewModeType.Details
    .DrawGridLines = True
    .Refresh()
End With
```

C++

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

    #import <ExShellView.dll>
    using namespace EXSHELLVIEWLib;
*/
EXSHELLVIEWLib::IExShellViewPtr spExShellView1 = GetDlgItem(IDC_EXSHELLVIEW1)-
>GetControlUnknown();
spExShellView1->PutViewMode(EXSHELLVIEWLib::Details);
spExShellView1->PutDrawGridLines(VARIANT_TRUE);
spExShellView1->Refresh();
```

C++ Builder

```
ExShellView1->ViewMode = Exshellviewlib_tlb::ViewModeType::Details;
ExShellView1->DrawGridLines = true;
```

```
ExShellView1->Refresh();
```

C#

```
exshellview1.ViewMode = exontrol.EXSHELLVIEWLib.ViewModeType.Details;
exshellview1.DrawGridLines = true;
exshellview1.Refresh();
```

JScript/JavaScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:B4E1F234-AF0D-4EAD-8113-A563B40E71CA"
id="ExShellView1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
   ExShellView1.ViewMode = 4;
   ExShellView1.DrawGridLines = true;
   ExShellView1.Refresh();
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload='Init()'>
<OBJECT CLASSID="clsid:B4E1F234-AF0D-4EAD-8113-A563B40E71CA"
id="ExShellView1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
   With ExShellView1
      .ViewMode = 4
      .DrawGridLines = True
```

```
      .Refresh
   End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
axExShellView1.ViewMode = EXSHELLVIEWLib.ViewModeType.Details;
axExShellView1.DrawGridLines = true;
axExShellView1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
   ;

   super();

   exshellview1.ViewMode(4/*Details*/);
   exshellview1.DrawGridLines(true);
   exshellview1.Refresh();
}
```

Delphi 8 (.NET only)

```
with AxExShellView1 do
begin
   ViewMode := EXSHELLVIEWLib.ViewModeType.Details;
   DrawGridLines := True;
   Refresh();
end
```

Delphi (standard)

```
with ExShellView1 do
```

```
begin
   ViewMode := EXSHELLVIEWLib_TLB.Details;
   DrawGridLines := True;
   Refresh();
end
```

VFP

```
with thisform.ExShellView1
   .ViewMode = 4
   .DrawGridLines = .T.
   .Refresh
endwith
```

dBASE Plus

```
local oExShellView

oExShellView = form.Activex1.nativeObject
oExShellView.ViewMode = 4
oExShellView.DrawGridLines = true
oExShellView.Refresh()
```

XBasic (Alpha Five)

```
Dim oExShellView as P

oExShellView = topparent:CONTROL_ACTIVEX1.activex
oExShellView.ViewMode = 4
oExShellView.DrawGridLines = .t.
oExShellView.Refresh()
```

Visual Objects

```

oDCOCX_Exontrol1:ViewMode := Details
oDCOCX_Exontrol1:DrawGridLines := true
```

```
oDCOCX_Exontrol1:Refresh()
```

PowerBuilder

```
OleObject oExShellView

oExShellView = ole_1.Object
oExShellView.ViewMode = 4
oExShellView.DrawGridLines = true
oExShellView.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
   Forward Send OnCreate
   Set ComViewMode to OLEDetails
   Set ComDrawGridLines to True
   Send ComRefresh
End_Procedure
```

XBase++

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

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

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

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

```
   oExShellView:CLSID  := "Exontrol.ShellView.1" /*{B4E1F234-AF0D-4EAD-8113-
A563B40E71CA}*/
   oExShellView:create(,, {10,60},{610,370} )

      oExShellView:ViewMode := 4/*Details*/
      oExShellView:DrawGridLines := .T.
      oExShellView:Refresh()

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

property ExShellView.Enabled as Boolean

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

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

This property determines if the control responds to user-generated events at run-time. The Enabled property allows ExShellView to be enabled or disabled at run-time. If control is disabled, no user-generated events are reported to ExShellView. Disabling ExShellView is possible, for display purposes, such as if you want only to provide read-only information.

property ExShellView.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 ExShellView.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 ExShellView1.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 ExShellView.FilePattern as String

Retrieves or sets a string value that indicates the file pattern used to include files. "*.jpg *.bmp "

Type	Description
String	A string expression that specifies the pattern that is used to filter files.

The FilePattern Property the enables user to filter out and show only specific items in the control. Typical wildcard expressions should be used to select a pattern. For example,

```
ExShellView1.FilePattern = "*.zip"
```

would display only items whose filename ends with ".zip' part (aka ZIP files). Also, any valid (wildcard) expression can be used. Please note that the FilePattern Property is valid only when the IncludeObjectType Property is set to the 'PatternObjects' constant.

property ExShellView.Font as IFontDisp

Retrieves or sets a Font object used to paint the items.

Type	Description
IFontDisp	A Font object being used to display items in the control.

Use the Font property to change the control's font.

property ExShellView.HeaderVisible as Boolean

Specifies whether the view's header is visible or hidden.

Type	Description
Boolean	A Boolean expression that specifies whether the control's header is visible or hidden.

By default, the HeaderVisible property is True. Use the HeaderVisible property to hide the control's header. The ViewMode property indicates the shell's view mode. The AutoArrange property specifies whether the files are automatically kept arranged in icon and small icon view. Use the AlignToGrid property to specifies whether in icon view, icons automatically snap into a grid. The HideFileNames property specifies whether the view shows the names for the files.

property ExShellView.HideFileNames as Boolean

Specifies whether the files hides the names in icon and small icon view.

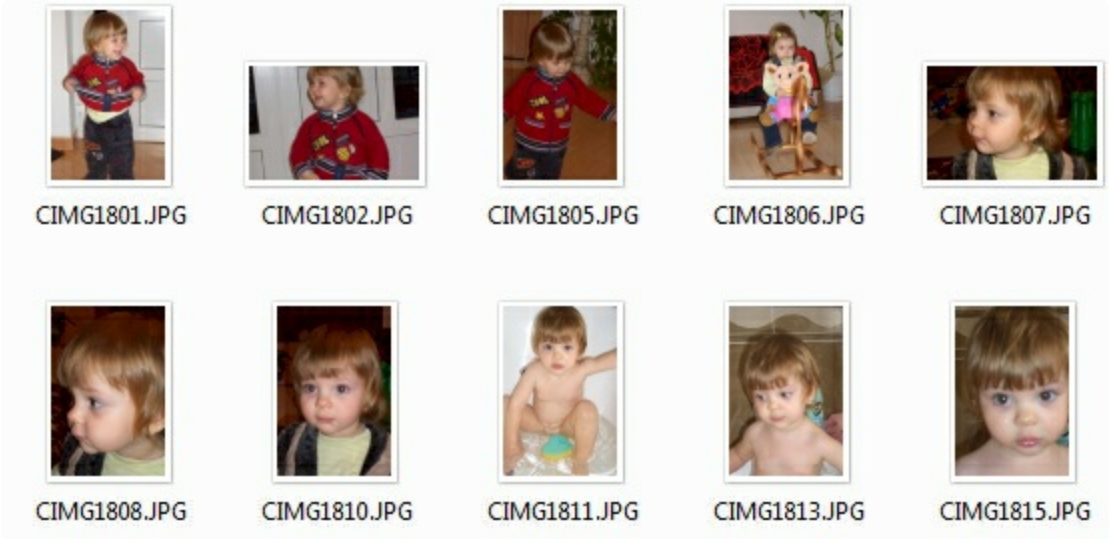
Type	Description
Boolean	A boolean expression that indicates whether the files hides the names in icon and small icon view.

By default, the HideFileNames property is False. Use the HideFileNames property to hide the names of the files in the view. The HideToolTips property specifies whether the control display a tooltip when the cursor hovers a file or folder. The HeaderVisible property specifies whether the shell displays its header (columns part on top side of the control)

This option may be available on newer version such as: Windows Vista, Windows 7, ...

The following screen shot shows a view with HideFileNames property is True:

and, the following screen shot shows a view with HideFileNames property is False,



property ExShellView.HideToolTips as Boolean

Specifies whether the file displays a tooltip when the cursor hovers the shellview.

Type	Description
Boolean	A Boolean expression that

By default, the HideToolTips property is True, which means that the tooltips are shown when the cursor hovers a file. Use the HideToolTips property to hide the tooltips whenever the cursor is hovering a file or folder. The HeaderVisible property specifies whether the shell displays its header (columns part on top side of the control) The ViewMode property indicates the shell's view mode. The AutoArrange property specifies whether the files are automatically kept arranged in icon and small icon view. Use the AlignToGrid property to specifies whether in icon view, icons automatically snap into a grid.

property ExShellView.hWnd as Long

Retrieves the 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 ExShellView.IncludeObjectType as IncludeObjectEnum

Retrieves or sets the way how the control will filter the objects.

Type	Description
IncludeObjectEnum	An IncludeObjectEnum expression that indicates the object being included in the list.

The IncludeObjectType property determines which files should be displayed on the screen when a folder is browsed. Depending on values showed below, it can show all objects, only folders, only user-defined objects (user determines which object should be shown run-time), and only ones that match given pattern with FilePattern property.

property ExShellView.MatchPattern (strPattern as String, strWord as String) as Boolean

Checks if the given word matches the given mask.

Type	Description
strPattern as String	A string expression that represents the pattern that will be used to test given string (ie. "*.*","?.exe" etc..
strWord as String	A string expression that is to be tested with given pattern
Boolean	A Boolean expression that specifies whether the specified word matches the giving pattern.

It might be pretty hard to test if certain text applies to a given pattern. Therefore, we included this method for you, so this test is done internally. This method will test if given string matches any pattern given by strPattern.

For example, this part of code will return True (tested in VB):

```
Debug.Print ExShellView1.MatchPattern("*.zip", "test.zip")
Debug.Print ExShellView1.MatchPattern("te*", "test.zip")
Debug.Print ExShellView1.MatchPattern("?es*", "test.zip")
Debug.Print ExShellView1.MatchPattern("t*p", "test.zip")
```

because 'test.zip' can by intentified by any of above patterns. Obviously, code

```
Debug.Print ExShellView1.MatchPattern("*.zip", "test.exe")
```

will return False, because this is not a valid match.

method ExShellView.ModifyFolderFlags (Add as FolderFlagsEnum, Remove as FolderFlagsEnum)

Adds or removes flags that indicates the options for browsed folder.

Type	Description
Add as FolderFlagsEnum	A combination of FolderFlagsEnum value that indicates the flags being added to current view.
Remove as FolderFlagsEnum	A combination of FolderFlagsEnum value that indicates the flags being removed from the current view.

This method determines custom flags that can apply to ExShellView determining its appearance. For example, reset the 'SingleSel' flag, and so the current view supports selecting multiple items. By default, the control supports selecting a single item. Use the ObjectSelected event to notify your application when the user selects an item. The ViewFloderFlags property is used to determine custom flags that are applied to the control determining its appearance. Use the SelectAll method to select all files in the control's view.

The following VB/NET sample shows how to enable multiple selection within the view:

```
Exshellview1.ViewFolderFlags = Exshellview1.ViewFolderFlags And Not
exontrol.EXSHELLVIEWLib.FolderFlagsEnum.SingleSel
```

The following C# sample shows how to enable multiple selection within the view:

```
exshellview1.ViewFolderFlags = exshellview1.ViewFolderFlags & ~
((int)exontrol.EXSHELLVIEWLib.FolderFlagsEnum.SingleSel);
```

The following VB.NET sample shows how to get the selected files/folder for /NET assembly version:

```
Dim i As Long = 0, s As String = ""
With Exshellview1
  .Objects.Get(exontrol.EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems)
  With .Objects
    For i = 0 To .Count - 1
      Dim sel As exontrol.EXSHELLVIEWLib.exshellobject = .Item(i)
      ' * The sel indicates the shell object being selected *
      s = s + sel.Name + vbCrLf
    Next
  End With
```

```
End With
If s.Length > 0 Then
   MessageBox.Show(s, "Selection")
Else
   MessageBox.Show("Empty", "Selection")
End If
```

The following C# sample shows how to get the selected files/folder for /NET assembly version:

```
string s = "";
exshellview1.Objects.Get(exontrol.EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems);
for ( int i = 0; i < exshellview1.Objects.Count; i++ )
{
   exontrol.EXSHELLVIEWLib.exshellobject sel = exshellview1.Objects[i];
   // * The sel indicates the shell object being selected *
   s = s + sel.Name + "\r\n";
}
if (s.Length > 0)
   MessageBox.Show(s, "Selection");
else
   MessageBox.Show("Empty", "Selection");
```

property ExShellView.Objects as ExShellObjects

Retrieves a collection of ExShellObject objects that indicates the current selection, or all items from the view.

Type	Description
ExShellObjects	An ExShellObjects collection that holds a collection of ExShellObject objects.

Use the Objects property to access the collection of selected items or all items. Use the Get method to fill the Objects collection with specified elements (selected or all items in the current view).

The Objects.Get method gets:

- nothing, if the objectType parameter is **NoItems**
- all files or folders being listed in the current view, if the objectType parameter is **AllItems**
- all files or folders being listed in the current view, as they are displayed, if the objectType parameter is **AllItems Or AsDisplayed**
- selected files or folders, if the objectType parameter is **SelectedItems**
- selected files or folders as they are displayed, if the objectType parameter is **SelectedItems or AsDisplayed**

The following VB6 sample gets a collection of selected items (in case your control allows multiple selection):

```
Private Sub ExShellView1_StateChange(ByVal newState As EXSHELLVIEWLibCtl.StatesEnum)
  If (newState = SelChangeState) Then
    ExShellView1.Objects.Get SelectedItems
    With ExShellView1.Objects
      For i = 0 To .Count - 1
        Debug.Print .Item(i).Name
      Next
    End With
  End If
End Sub
```

In case your control supports single selection, you can use the ObjectSelected event to

notify when a new item/object is selected:

```
Private Sub ExShellView1_ObjectSelected(ByVal Object As
EXSHELLVIEWLibCtl.IExShellObject)
   If Not (Object Is Nothing) Then
      Debug.Print Object.Name
   End If
End Sub
```

The following C++ sample displays a message box with the Name of all selected files and folders:

```
#import <ExShellView.dll>
using namespace EXSHELLVIEWLib;

void GetSelectedObjects( EXSHELLVIEWLib::IExShellView* pShellView )
{
   pShellView->GetObjects()->Get(EXSHELLVIEWLib::SelectedItems);
   EXSHELLVIEWLib::IExShellObjectsPtr spObjects = pShellView->GetObjects();
   for ( long i = 0; i < spObjects->Count; i++ )
      ::MessageBox( NULL, spObjects->GetItem( i )->Name, NULL, NULL );
}
```

The following VB.NET sample shows how to get the selected files/folder for /NET assembly version:

```
Dim i As Long = 0, s As String = ""
With Exshellview1
   .Objects.Get(exontrol.EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems)
   With .Objects
      For i = 0 To .Count - 1
         Dim sel As exontrol.EXSHELLVIEWLib.exshellobject = .Item(i)
         ' * The sel indicates the shell object being selected *
         s = s + sel.Name + vbCrLf
      Next
   End With
End With
If s.Length > 0 Then
   MessageBox.Show(s, "Selection")
```

```
Else
   MessageBox.Show("Empty", "Selection")
End If
```

The following C# sample shows how to get the selected files/folder for /NET assembly version:

```
string s = "";
exshellview1.Objects.Get(exontrol.EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems);
for ( int i = 0; i < exshellview1.Objects.Count; i++ )
{
   exontrol.EXSHELLVIEWLib.exshellobject sel = exshellview1.Objects[i];
   // * The sel indicates the shell object being selected *
   s = s + sel.Name + "\r\n";
}
if (s.Length > 0)
   MessageBox.Show(s, "Selection");
else
   MessageBox.Show("Empty", "Selection");
```

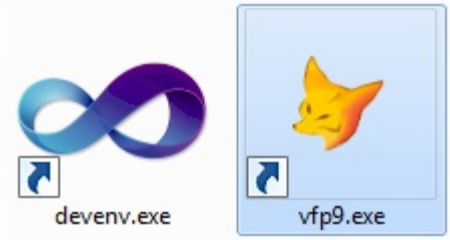
property ExShellView.OverlayIcons as Boolean

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

Type	Description
Boolean	A Boolean expression that specifies whether the control shows or hides the overlay icons.

By default, the OverlayIcons property is True, which indicates that the control displays the overlay icons. 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. The ViewMode property indicates the way the shell displays the files/folders inside.

The following screen shot shows the shell view with overlay icons:



while the next screen shot shows the shell view with no overlay icons:

method ExShellView.Refresh ()

Refreshes the content of the browsed folder.

Type	Description

Forces reloading the current view as well as refreshing the current view.

property ExShellView.SelectAll as Boolean

Selects or unselects all files in the control when the multiple selection is enabled.

Type	Description
Boolean	A Boolean expression that specifies whether the method selects all items (true) or unselect all items (false)

The SelectAll property is write only. The SelectAll on True, selects all items, while the SelectAll on False, unselects all items in the control's view. The SelectAll property has effect only if the control allows multiple selection. The ViewFloderFlags property is used to determine custom flags that are applied to the control determining its appearance. Use the ModifyFolderFlags method to add or remove flags on the current view, including enabling single or multiple selection.

The following VB/NET sample shows how to enable multiple selection within the view:

```
Exshellview1.ViewFolderFlags = Exshellview1.ViewFolderFlags And Not
exontrol.EXSHELLVIEWLib.FolderFlagsEnum.SingleSel
```

The following C# sample shows how to enable multiple selection within the view:

```
exshellview1.ViewFolderFlags = exshellview1.ViewFolderFlags & ~
((int)exontrol.EXSHELLVIEWLib.FolderFlagsEnum.SingleSel);
```

The following VB.NET sample shows how to get the selected files/folder for /NET assembly version:

```
Dim i As Long = 0, s As String = ""
With Exshellview1
   .Objects.Get(exontrol.EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems)
   With .Objects
      For i = 0 To .Count - 1
         Dim sel As exontrol.EXSHELLVIEWLib.exshellobject = .Item(i)
         ' * The sel indicates the shell object being selected *
         s = s + sel.Name + vbCrLf
      Next
   End With
End With
If s.Length > 0 Then
   MessageBox.Show(s, "Selection")
```

```
Else
   MessageBox.Show("Empty", "Selection")
End If
```

The following C# sample shows how to get the selected files/folder for /NET assembly version:

```
string s = "";
exshellview1.Objects.Get(exontrol.EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems);
for ( int i = 0; i < exshellview1.Objects.Count; i++ )
{
   exontrol.EXSHELLVIEWLib.exshellobject sel = exshellview1.Objects[i];
   // * The sel indicates the shell object being selected *
   s = s + sel.Name + "\r\n";
}
if (s.Length > 0)
   MessageBox.Show(s, "Selection");
else
   MessageBox.Show("Empty", "Selection");
```

property ExShellView.ShellFolder (Path as Variant) as ExShellFolder

Retrieves a Folder object based on a path, on a special folder constant or on an ID property.

Type	Description
Path as Variant	A string expression that represents a path of folder whose IShellFolder is needed.
ExShellFolder	An ExShellFolder object being created based on the path.

This property is mostly used for creating ExShellView's ExShellFolder. All of ExShellView's browsing strategy is based on such objects. Since we're dealing with the objects here, it is not enough just to specify Path to be set for browsing. This property helps us to generate appropriate object based on a given path. The SpecialFolder property indicates a common folder in your Windows.

Therefore, instead of doing something like this

```
ExShellView1.Path = "C:\WINDOWS" ' Bad!
```

you should write this line

```
ExShellView1.BrowseFolder = ExShellView1.ShellFolder("C:\WINDOWS")
```

so, ShellFolder property created ExShellFolder object for us based on a path, and we used that object to set new folder using BrowseFolder property

property ExShellView.SpecialFolder (SpecialFolder as SpecialFolderPathConstants) as ExShellFolder

Retrieves a Folder object given a special folder constant.

Type	Description
SpecialFolder as SpecialFolderPathConstants	A constant value that is used to determine path so specific shell folder
ExShellFolder	An ExShellFolder object that specifies the special folder.

Windows OS has several folders that are called 'shell folders'. For example, those are 'Program files','Recycle Bin', etc. Since these folders may be different among different PC's, this property is used to determine path to such folders on a local computer. Using this value, instead of constant path ensures portability of your software among different Windows OS'es. Use the BrowseFolder property to browse for another folder.

property ExShellView.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\ExExShellView\Sample\SAMPLE.MDB", 3, 3 )
}
DataSource = rs
```

property ExShellView.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 ExShellView.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.*

method ExShellView.UpOneLevel ()

Browses the parent of current browsed folder.

Type	Description

This method browses the parent folder. As in most Explorer-based applications, there is a 'Up one level' button that sets parent folder as current, this method does the same. For example, if 'My Computer' folder is currently browsed, using this method will result in setting 'Desktop' folder as browsed.

property ExShellView.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 ExShellView.ViewFolderFlags as Long

Retrieves the flags for the browsed folder.

Type	Description
Long	A Long expression that indicates the flags that determines the control's appearance. This value is a combination of FolderFlagsEnum values.

This property is used to determine custom flags that are applied to the control determining its appearance. You can use the ModifyFolderFlags method to add or remove specified flags to the current view. The most common option is single or multiple selection.

The following VB/NET sample shows how to enable multiple selection within the view (/NET Assembly):

```
Exshellview1.ViewFolderFlags = Exshellview1.ViewFolderFlags And Not
exontrol.EXSHELLVIEWLib.FolderFlagsEnum.SingleSel
```

The following C# sample shows how to enable multiple selection within the view (/NET Assembly):

```
exshellview1.ViewFolderFlags = exshellview1.ViewFolderFlags & ~
((int)exontrol.EXSHELLVIEWLib.FolderFlagsEnum.SingleSel);
```

The following VB/NET sample shows how to enable multiple selection within the view (/COM on Window.Forms):

```
AxExShellView1.ViewFolderFlags = AxExShellView1.ViewFolderFlags And Not
EXSHELLVIEWLib.FolderFlagsEnum.SingleSel
```

The following C# sample shows how to enable multiple selection within the view (/COM on Window.Forms):

```
axExShellView1.ViewFolderFlags = axExShellView1.ViewFolderFlags & ~
((int)EXSHELLVIEWLib.FolderFlagsEnum.SingleSel);
```

The following VFP sample shows how to enable multiple selection within the view (/COM on Window.Forms):

```
thisform.Exshellview1.ViewFolderFlags = bitand(thisform.Exshellview1.ViewFolderFlags,
bitnot(64) )
```

The following C++ sample shows how to enable multiple selection within the view (/COM on Window.Forms):

```
m_shellView.SetViewFolderFlags( m_shellView.GetViewFolderFlags() & ~64 );
```

The following VB.NET sample shows how to get the selected files/folder for /NET assembly version:

```
Dim i As Long = 0, s As String = ""
With Exshellview1
   .Objects.Get(exontrol.EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems)
   With .Objects
      For i = 0 To .Count - 1
         Dim sel As exontrol.EXSHELLVIEWLib.exshellobject = .Item(i)
         ' * The sel indicates the shell object being selected *
         s = s + sel.Name + vbCrLf
      Next
   End With
End With
If s.Length > 0 Then
   MessageBox.Show(s, "Selection")
Else
   MessageBox.Show("Empty", "Selection")
End If
```

The following C# sample shows how to get the selected files/folder for /NET assembly version:

```
string s = "";
exshellview1.Objects.Get(exontrol.EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems);
for ( int i = 0; i < exshellview1.Objects.Count; i++ )
{
   exontrol.EXSHELLVIEWLib.exshellobject sel = exshellview1.Objects[i];
   // * The sel indicates the shell object being selected *
   s = s + sel.Name + "\r\n";
}
if (s.Length > 0)
   MessageBox.Show(s, "Selection");
else
```

```
MessageBox.Show("Empty", "Selection");
```

property ExShellView.ViewMode as ViewModeType

Returns or changes the current view mode of the control.

Type	Description
ViewModeType	A constant value that determine the view mode for the current folder.

As in standard Explorer, there are several commonly used viewmode's for representing folder's objects. Mostly is used 'Large Icons', but there are also other modes, as SmallIcon, List, Details or Thumbnail. On Windows Vista, Windows 7, there are also Extra Large Icons, Large Icons and Medium Icons view modes. The AutoArrange property specifies whether the files are automatically kept arranged in icon and small icon view. Use the AlignToGrid property to specifies whether in icon view, icons automatically snap into a grid. The HideFileNames property specifies whether the view shows the names for the files. The Objects.Get(SelectedItems) property to get a collection of selected files/folders. The DrawGridLines property specifies whether the control shows or hides the grid lines around the files/folders.

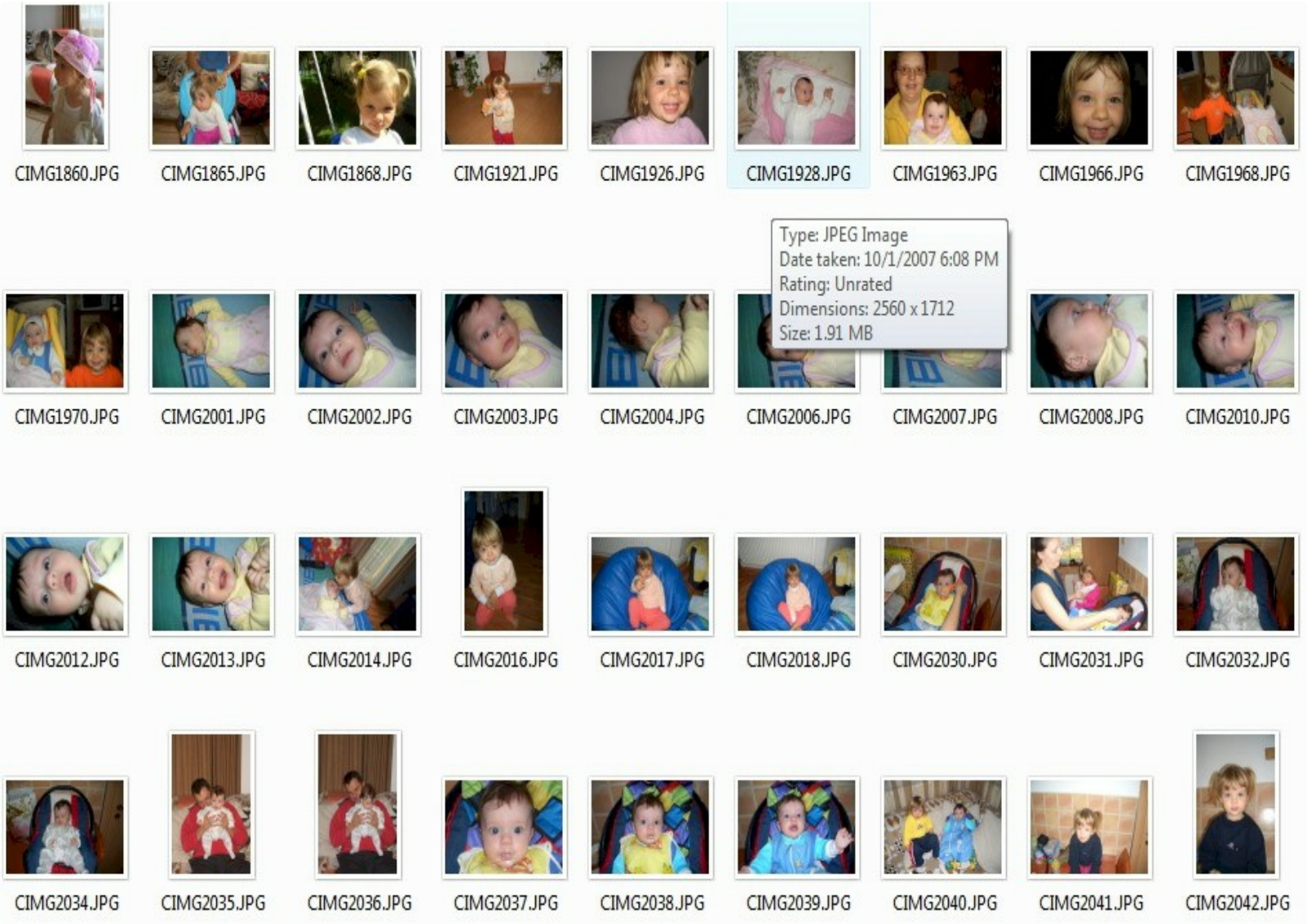
The view's ***Extra_Large_Icons*** mode may shows as:

CIMG1860.JPG

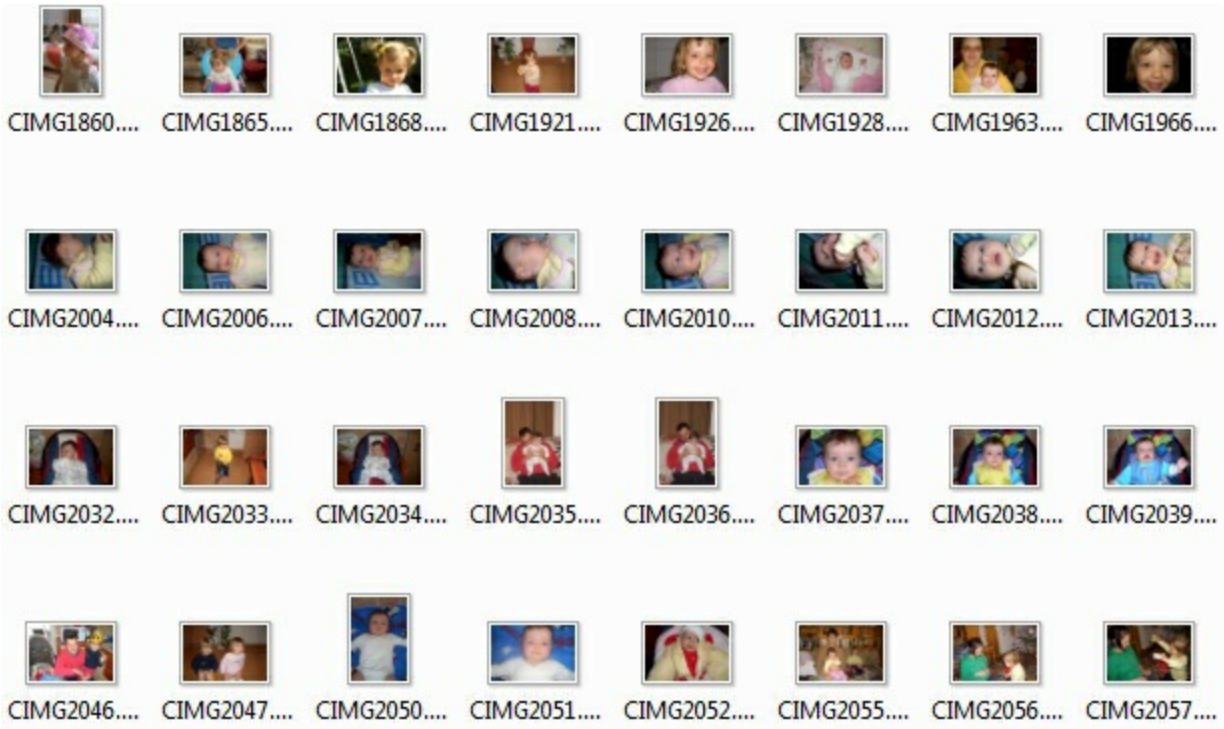
CIMG1865.JPG

CIMG1868.JPG

The view's ***Large_Icons*** mode may shows as:



The view's ***Medium_Icons*** mode may shows as:



ExShellView events

Tip The /COM object can be placed on a HTML page (with usage of the HTML object tag: <object classid="clsid:...">) using the class identifier: {B4E1F234-AF0D-4EAD-8113-A563B40E71CA}. The object's program identifier is: "Exontrol.ShellView". The /COM object module is: "ExShellView.dll"

The eXShellView supports the following events:

Name	Description
BrowseFolderChange	Fired after a new folder was browsed.
Click	Occurs when the user presses and then releases the left mouse button over the tree control.
DblClick	Occurs when the user dblclk the left mouse button over an object.
IncludeObject	Fired during loading items, to filter the item objects.
InvokeItemMenu	Notifies the application once the user selects a command in the context menu.
InvokeMenuCommand	Fired when the user selects an item context menu that has been added during QueryContextMenu event.
KeyDown	Occurs when the user presses a key while an object has the focus.
KeyPress	Occurs when the user presses and releases an ANSI key.
KeyUp	Occurs when the user releases a key while an object has the focus.
ObjectSelect	Fired when the user selects a new object for browsing.
ObjectSelected	Fired when a new object was selected.
QueryContextMenu	Fired when the context menu is about to be active. You can supply new items to the context menu.
StateChange	Fired when the list's state has been changed: focus, selection.

event BrowseFolderChange ()

Fired after a new folder was browsed.

Type	Description

It indicates that the control's changed the currently browsed folder. Use the CurrentFolder / BrowseFolder property to get the current folder. The programmer might put some initialization code here, or update it's variables, etc.

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

C#

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

VB

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

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

C#

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

C++

```
void OnBrowseFolderChange()
{
}
```

C++ Builder

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

Delphi

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

Delphi 8 (.NET only)

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

Powe…

```
begin event BrowseFolderChange()
end event BrowseFolderChange
```

VB.NET

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

VB6

```
Private Sub BrowseFolderChange()
End Sub
```

VBA

```
Private Sub BrowseFolderChange()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnBrowseFolderChange(oExShellView)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComBrowseFolderChange
   Forward Send OnComBrowseFolderChange
End_Procedure
```

Visual Objects

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

X++

```
void onEvent_BrowseFolderChange()
{
```

```
}
```

XBasic

```
function BrowseFolderChange as v ()
end function
```

dBASE

```
function nativeObject_BrowseFolderChange()
return
```

For instance, the following VB6 sample changes the browsing path for ExFolderView and ExFolderCombo components:

```
Private Sub ExShellView1_BrowseFolderChange()
   ExFolderCombo1.OpenedFolder = ExShellView1.BrowseFolder
   ExFolderView1.SelectedFolder = ExShellView1.BrowseFolder
End Sub
```

For instance, the following VB.NET displays the name of browsed folder:

```
Private Sub Exshellview1_BrowseFolderChange(ByVal sender As System.Object) Handles
Exshellview1.BrowseFolderChange
   With Exshellview1
      Dim b As exontrol.EXSHELLVIEWLib.exshellfolder = .BrowseFolder
      Debug.Print("Exshellview1_BrowseFolderChange " & b.Path)
   End With
End Sub
```

event Click ()

Fired when the user clicks on the control area.

Type	Description

The Click event is fired when the user releases the left mouse button over the control. You can use the QueryContextMenu event to be notified when the user right clicks the control's view. The Objects property specifies the collection of all or selected files/folders in the control. Use the Get method to update the Objects collection with all or selected files or folders, and then you can enumerate the files in the collection using the Count and Item properties.

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(oExShellView)
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
```

The following VB sample displays the file/folder being clicked:

```
Private Sub ExShellView1_Click()
  With ExShellView1
    .Objects.Get (SelectedItems Or AsDisplayed)
    With .Objects
      Dim i As Long
      For i = 0 To .Count - 1
        Debug.Print .Item(i).Path
      Next
    End With
  End With
End Sub
```

event DblClick ()

Fired when the user dblclicks on the control area.

Type	Description

The DblClick event notifies your application when the user double clicks an object in the shell view control. The Objects property specifies the collection of all or selected files/folders in the control. Use the Get method to update the Objects collection with all or selected files or folders, and then you can enumerate the files in the collection using the Count and Item properties.

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(oExShellView)
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
```

The following VB6 sample opens the file being double clicked.

```
Private Sub ExShellView1_DblClick()
  With ExShellView1
    .Objects.Get (SelectedItems)
    With ExShellView1.Objects
      If (.Count > 0) Then
        Dim i As EXSHELLVIEWLibCtl.ExShellObject
        Set i = .Item(0)
        If (Not i.Attribute(IsFolder)) Then
          i.InvokeCommand ("Open")
        End If
      End If
    End With
  End With
End Sub
```

event IncludeObject (Object as Object, Include as Variant)

Fired during loading items, to filter the item objects.

Type	Description
Object as Object	Reference to ExShellObject that should or shouldn't be visible.
Include as Variant	A boolean expression that makes object visible (True), or hides it (False).

When user selects some folder to browse, he also sets type of files that needs to be shown. Usually, this is done using FilePattern property, and IncludeObjectType property. This event is fired only if user set IncludeObjectType property to value 'UserObjects'. Each time eXShellView needs to determine if particular item should be visible or not, this event is fired. User should set Include variable to either True, or False, depending if he chooses to show or hide that item.

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

C#
```
private void IncludeObject(object sender,object Obj,ref object Include)
{
}
```

VB
```
Private Sub IncludeObject(ByVal sender As System.Object,ByVal Obj As
Object,ByRef Include As Object) Handles IncludeObject
End Sub
```

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

C#
```
private void IncludeObject(object sender,
AxEXSHELLVIEWLib._IExShellViewEvents_IncludeObjectEvent e)
{
}
```

C++
```
void OnIncludeObject(LPDISPATCH Object,VARIANT FAR* Include)
{
}
```

C++ Builder
```
void __fastcall IncludeObject(TObject *Sender,IDispatch *Object,Variant * Include)
{
```

```
}
```

Delphi

```
procedure IncludeObject(ASender: TObject; Object : IDispatch;var Include :
OleVariant);
begin
end;
```

Delphi 8 (.NET only)

```
procedure IncludeObject(sender: System.Object; e:
AxEXSHELLVIEWLib._IExShellViewEvents_IncludeObjectEvent);
begin
end;
```

Powe…

```
begin event IncludeObject(oleobject Object,any Include)
end event IncludeObject
```

VB.NET

```
Private Sub IncludeObject(ByVal sender As System.Object, ByVal e As
AxEXSHELLVIEWLib._IExShellViewEvents_IncludeObjectEvent) Handles
IncludeObject
End Sub
```

VB6

```
Private Sub IncludeObject(ByVal Object As Object,Include As Variant)
End Sub
```

VBA

```
Private Sub IncludeObject(ByVal Object As Object,Include As Variant)
End Sub
```

VFP

```
LPARAMETERS Object,Include
```

Xbas…

```
PROCEDURE OnIncludeObject(oExShellView,Object,Include)
RETURN
```

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

Java…

```
<SCRIPT EVENT="IncludeObject(Object,Include)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

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

```
Function IncludeObject(Object,Include)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComIncludeObject Variant llObject Variant llInclude
   Forward Send OnComIncludeObject llObject llInclude
End_Procedure
```

Visual Objects

```
METHOD OCX_IncludeObject(Object,Include) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_IncludeObject(COM _Object,COMVariant /*variant*/ _Include)
{
}
```

XBasic

```
function IncludeObject as v (Object as P,Include as A)
end function
```

dBASE

```
function nativeObject_IncludeObject(Object,Include)
return
```

event InvokeItemMenu (Command as Long)

Notifies the application once the user selects a command in the context menu.

Type	Description
Command as Long	A Long expression that indicates the identifier of the command being selected.

The InvokeItemMenu event notifies the application once the user selects an item from the control's context menu. For instance, you can use the InvokeItemMenu event to be notified when the user changes the view mode. The ViewMode property determines the current view mode. Use the InvokeMenuCommand event to be notifies when the user selects a custom command that previously was added using the QueryContextMenu event.

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

C#

```
private void InvokeItemMenu(object sender,int Command)
{
}
```

VB

```
Private Sub InvokeItemMenu(ByVal sender As System.Object,ByVal Command As
Integer) Handles InvokeItemMenu
End Sub
```

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

C#

```
private void InvokeItemMenu(object sender,
AxEXSHELLVIEWLib._IExShellViewEvents_InvokeItemMenuEvent e)
{
}
```

C++

```
void OnInvokeItemMenu(long Command)
{
}
```

C++ Builder

```
void __fastcall InvokeItemMenu(TObject *Sender,long Command)
{
}
```

Delphi

```
procedure InvokeItemMenu(ASender: TObject; Command : Integer);
begin
```

```
end;
```

Delphi 8 (.NET only)

```
procedure InvokeItemMenu(sender: System.Object; e:
AxEXSHELLVIEWLib._IExShellViewEvents_InvokeItemMenuEvent);
begin
end;
```

Powe…

```
begin event InvokeItemMenu(long Command)
end event InvokeItemMenu
```

VB.NET

```
Private Sub InvokeItemMenu(ByVal sender As System.Object, ByVal e As
AxEXSHELLVIEWLib._IExShellViewEvents_InvokeItemMenuEvent) Handles
InvokeItemMenu
End Sub
```

VB6

```
Private Sub InvokeItemMenu(ByVal Command As Long)
End Sub
```

VBA

```
Private Sub InvokeItemMenu(ByVal Command As Long)
End Sub
```

VFP

```
LPARAMETERS Command
```

Xbas…

```
PROCEDURE OnInvokeItemMenu(oExShellView,Command)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComInvokeItemMenu Integer llCommand
   Forward Send OnComInvokeItemMenu llCommand
End_Procedure
```

Visual Objects

```
METHOD OCX_InvokeItemMenu(Command) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_InvokeItemMenu(int _Command)
{
}
```

XBasic

```
function InvokeItemMenu as v (Command as N)
end function
```

dBASE

```
function nativeObject_InvokeItemMenu(Command)
return
```

The following VB sample displays a message box when the user changes the view mode:

```
Dim nVM As EXSHELLVIEWLibCtl.ViewModeType

Private Sub ExShellView1_InvokeItemMenu(ByVal Command As Long)

   If (nVM <> ExShellView1.ViewMode) Then
      nVM = ExShellView1.ViewMode
      MsgBox "ViewMode changed to " & nVM
   End If

End Sub

Private Sub Form_Load()
   nVM = ExShellView1.ViewMode
End Sub
```

The sample holds the current view mode when the application starts on nVM variable. Once the InvokeItemMenu event occurs, the nVM variable is checked with the current view mode,

and if it changed, a message box is displayed that the user has changed the view mode.

The following C# sample displays a message box once the user changes the control's view mode:

```
public Form1()
{
   InitializeComponent();

   nVM = exshellview1.ViewMode;
}

exontrol.EXSHELLVIEWLib.ViewModeType nVM =
exontrol.EXSHELLVIEWLib.ViewModeType.Details;
private void exshellview1_InvokeItemMenu(object sender, int Command)
{
   if ( nVM != exshellview1.ViewMode )
   {
      nVM = exshellview1.ViewMode;
      MessageBox.Show( "ViewMode changed to " + nVM.ToString());
   }
}
```

The Values for the Command parameter are determined by the system, and are the same for any Windows version. For instance, the 30995 indicates a Rename operation, while the 30996 command invokes the Properties dialog of selected file or folder.

The following VB sample displays the command being performed:

```
Private Sub ExShellView1_InvokeItemMenu(ByVal Command As Long)
   Debug.Print Command
End Sub
```

event InvokeMenuCommand (Command as String)

Fired when the user selects an item context menu that has been added during QueryContextMenu event.

Type	Description
Command as String	A String expression that indicates the caption of the custom command being executed.

Use the InvokeMenuCommand event to notify your application when the user selects a custom command, that was previously added using the QueryContextMenu event. The InvokeItemMenu event notifies the application once the user selects a command by identifier in the context menu. Use the DefaultMenuItems property to specify whether the context menu shows the default menu items. The InvokeMenuCommand event is not fired if user clicks a default command. It is fired only for items being added using the QueryContextMenu event.

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

C#

```
private void InvokeMenuCommand(object sender,string Command)
{
}
```

VB

```
Private Sub InvokeMenuCommand(ByVal sender As System.Object,ByVal
Command As String) Handles InvokeMenuCommand
End Sub
```

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

C#

```
private void InvokeMenuCommand(object sender,
AxEXSHELLVIEWLib._IExShellViewEvents_InvokeMenuCommandEvent e)
{
}
```

C++

```
void OnInvokeMenuCommand(LPCTSTR Command)
{
}
```

C++ Builder

```
void __fastcall InvokeMenuCommand(TObject *Sender,BSTR Command)
{
}
```

Delphi

```
procedure InvokeMenuCommand(ASender: TObject; Command : WideString);
begin
end;
```

Delphi 8 (.NET only)

```
procedure InvokeMenuCommand(sender: System.Object; e:
AxEXSHELLVIEWLib._IExShellViewEvents_InvokeMenuCommandEvent);
begin
end;
```

Powe…

```
begin event InvokeMenuCommand(string Command)
end event InvokeMenuCommand
```

VB.NET

```
Private Sub InvokeMenuCommand(ByVal sender As System.Object, ByVal e As
AxEXSHELLVIEWLib._IExShellViewEvents_InvokeMenuCommandEvent) Handles
InvokeMenuCommand
End Sub
```

VB6

```
Private Sub InvokeMenuCommand(ByVal Command As String)
End Sub
```

VBA

```
Private Sub InvokeMenuCommand(ByVal Command As String)
End Sub
```

VFP

```
LPARAMETERS Command
```

Xbas…

```
PROCEDURE OnInvokeMenuCommand(oExShellView,Command)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComInvokeMenuCommand String llCommand
    Forward Send OnComInvokeMenuCommand llCommand
End_Procedure
```

Visual Objects

```
METHOD OCX_InvokeMenuCommand(Command) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_InvokeMenuCommand(str _Command)
{
}
```

XBasic

```
function InvokeMenuCommand as v (Command as C)
end function
```

dBASE

```
function nativeObject_InvokeMenuCommand(Command)
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 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,
AxEXSHELLVIEWLib._IExShellViewEvents_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:
AxEXSHELLVIEWLib._IExShellViewEvents_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
AxEXSHELLVIEWLib._IExShellViewEvents_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(oExShellView,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 displays the object's Properties dialog, when the user presses the F2 key:

```
Private Sub ExShellView1_KeyDown(KeyCode As Integer, Shift As Integer)
    If KeyCode = vbKeyF2 Then
        ExShellView1.Objects.Get (SelectedItems)
        With ExShellView1.Objects
            If (.Count > 0) Then
                .Item(0).InvokeCommand ("Properties")
            End If
        End With
```

```
    End If
End Sub
```

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

```
Private Sub ExShellView1_KeyDown(KeyCode As Integer, Shift As Integer)
    If KeyCode = vbKeyF2 Then
        ExShellView1.Objects.Get (SelectedItems)
        With ExShellView1.Objects
            If (.Count > 0) Then
                .Item(0).InvokeRename
            End If
        End With
    End If
End Sub
```

event KeyPress (KeyAscii as Integer)

Fired when an user presses a key.

Type	Description
KeyAscii as Integer	An integer value that represents the ASCII code for a pressed key.

Every time user pressed any key, this event is fired. KeyAscii variable holds ASCII value of pressed.

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,
AxEXSHELLVIEWLib._IExShellViewEvents_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:
AxEXSHELLVIEWLib._IExShellViewEvents_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
AxEXSHELLVIEWLib._IExShellViewEvents_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(oExShellView,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,
AxEXSHELLVIEWLib._IExShellViewEvents_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:
AxEXSHELLVIEWLib._IExShellViewEvents_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
AxEXSHELLVIEWLib._IExShellViewEvents_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(oExShellView,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 ObjectSelect (Object as ExShellObject)

Fired when the user selects a new object for browsing.

Type	Description
Object as ExShellObject	A reference to the ExShellObject being selected.

This event is fired when the user double-clicks or presses Enter key on any object in the browser. By default, if a folder item is being double clicked, the folder gets browsed. If a file is being double clicked, nothing is happen. Use the CancelObjectSelect method to prevent opening or browsing the folder being double clicked (or any other type of object).

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

C#

```
private void ObjectSelect(object sender,exontrol.EXSHELLVIEWLib.ExShellObject
Obj)
{
}
```

VB

```
Private Sub ObjectSelect(ByVal sender As System.Object,ByVal Obj As
exontrol.EXSHELLVIEWLib.ExShellObject) Handles ObjectSelect
End Sub
```

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

C#

```
private void ObjectSelect(object sender,
AxEXSHELLVIEWLib._IExShellViewEvents_ObjectSelectEvent e)
{
}
```

C++

```
void OnObjectSelect(LPDISPATCH Object)
{
}
```

C++ Builder

```
void __fastcall ObjectSelect(TObject *Sender,Exshellviewlib_tlb::IExShellObject
*Object)
{
}
```

Delphi

```
procedure ObjectSelect(ASender: TObject; Object : IExShellObject);
```

```
begin
end;
```

Delphi 8 (.NET only)

```
procedure ObjectSelect(sender: System.Object; e:
AxEXSHELLVIEWLib._IExShellViewEvents_ObjectSelectEvent);
begin
end;
```

Powe…

```
begin event ObjectSelect(oleobject Object)
end event ObjectSelect
```

VB.NET

```
Private Sub ObjectSelect(ByVal sender As System.Object, ByVal e As
AxEXSHELLVIEWLib._IExShellViewEvents_ObjectSelectEvent) Handles ObjectSelect
End Sub
```

VB6

```
Private Sub ObjectSelect(ByVal Object As EXSHELLVIEWLibCtl.IExShellObject)
End Sub
```

VBA

```
Private Sub ObjectSelect(ByVal Object As Object)
End Sub
```

VFP

```
LPARAMETERS Object
```

Xbas…

```
PROCEDURE OnObjectSelect(oExShellView,Object)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComObjectSelect Variant llObject
   Forward Send OnComObjectSelect llObject
End_Procedure
```

Visual Objects

```
METHOD OCX_ObjectSelect(Object) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_ObjectSelect(COM _Object)
{
}
```

XBasic

```
function ObjectSelect as v (Object as OLE::Exontrol.ShellView.1::IExShellObject)
end function
```

dBASE

```
function nativeObject_ObjectSelect(Object)
return
```

The following VB sample prevents opening a zip file (which is considered a folder item):

```
Private Sub ExShellView1_ObjectSelect(ByVal Object As
EXSHELLVIEWLibCtl.IExShellObject)
   If Object.Name Like "*.zip" Then
      ExShellView1.CancelObjectSelect
   End If
End Sub
```

event ObjectSelected (Object as ExShellObject)

Fired when a new object is selected.

Type	Description
Object as ExShellObject	A reference to the ExShellObject being selected.

The ObjectSelected event notifies your application that a new items in the list is selected. The StateChange event notifies your application whether the control loses or gains the focus, when the user renamed an item, or whether the user selects a new item.

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

C#

```
private void ObjectSelected(object sender,exontrol.EXSHELLVIEWLib.ExShellObject Obj)
{
}
```

VB

```
Private Sub ObjectSelected(ByVal sender As System.Object,ByVal Obj As exontrol.EXSHELLVIEWLib.ExShellObject) Handles ObjectSelected
End Sub
```

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

C#

```
private void ObjectSelected(object sender, AxEXSHELLVIEWLib._IExShellViewEvents_ObjectSelectedEvent e)
{
}
```

C++

```
void OnObjectSelected(LPDISPATCH Object)
{
}
```

C++ Builder

```
void __fastcall ObjectSelected(TObject *Sender,Exshellviewlib_tlb::IExShellObject *Object)
{
}
```

Delphi

```
procedure ObjectSelected(ASender: TObject; Object : IExShellObject);
begin
```

```
end;
```

Delphi 8 (.NET only)

```
procedure ObjectSelected(sender: System.Object; e:
AxEXSHELLVIEWLib._IExShellViewEvents_ObjectSelectedEvent);
begin
end;
```

Powe…

```
begin event ObjectSelected(oleobject Object)
end event ObjectSelected
```

VB.NET

```
Private Sub ObjectSelected(ByVal sender As System.Object, ByVal e As
AxEXSHELLVIEWLib._IExShellViewEvents_ObjectSelectedEvent) Handles
ObjectSelected
End Sub
```

VB6

```
Private Sub ObjectSelected(ByVal Object As EXSHELLVIEWLibCtl.IExShellObject)
End Sub
```

VBA

```
Private Sub ObjectSelected(ByVal Object As Object)
End Sub
```

VFP

```
LPARAMETERS Object
```

Xbas…

```
PROCEDURE OnObjectSelected(oExShellView,Object)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComObjectSelected Variant llObject
   Forward Send OnComObjectSelected llObject
End_Procedure
```

Visual Objects

```
METHOD OCX_ObjectSelected(Object) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_ObjectSelected(COM _Object)
{
}
```

XBasic

```
function ObjectSelected as v (Object as OLE::Exontrol.ShellView.1::IExShellObject)
end function
```

dBASE

```
function nativeObject_ObjectSelected(Object)
return
```

In case your control supports single selection, you can use the ObjectSelected event to notify when a new item/object is selected:

```
Private Sub ExShellView1_ObjectSelected(ByVal Object As
EXSHELLVIEWLibCtl.IExShellObject)
   If Not (Object Is Nothing) Then
      Debug.Print Object.Name
   End If
End Sub
```

The following sample gets a collection of selected items (in case your control allows multiple selection):

```
Private Sub ExShellView1_StateChange(ByVal newState As EXSHELLVIEWLibCtl.StatesEnum)
   If (newState = SelChangeState) Then
      ExShellView1.Objects.Get SelectedItems
      With ExShellView1.Objects
         For i = 0 To .Count - 1
            Debug.Print .Item(i).Name
```

```
        Next
    End With
  End If
End Sub
```

event QueryContextMenu (Items as String, Separator as String)

Fired when the context menu is about to be shown.

Type	Description
Items as String	A String expression that indicates the list of custom commands being added.
Separator as String	A String expression that indicates the separator for the list of commands in the Items parameter.

Use the QueryContextMenu to supply new items to the control's context menu. Use the QueryContextMenu event to display your own popup/context menu. The DefaultMenuItems property specifies whether the control displays the default context menu. The InvokeMenuCommand event notifies your application that the user selects a custom command. If you need to provide your own context menu, set the DefaultMenuItems property on False, and handle the QueryContextMenu event when your context menu to be shown.

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

C#

```
private void QueryContextMenu(object sender,ref string Items,ref string Separator)
{
}
```

VB

```
Private Sub QueryContextMenu(ByVal sender As System.Object,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, AxEXSHELLVIEWLib._IExShellViewEvents_QueryContextMenuEvent e)
{
}
```

C++

```
void OnQueryContextMenu(LPCTSTR FAR* Items,LPCTSTR FAR* Separator)
{
}
```

C++ Builder

```
void __fastcall QueryContextMenu(TObject *Sender,BSTR * Items,BSTR * Separator)
{
}
```

Delphi

```
procedure QueryContextMenu(ASender: TObject; var Items : WideString;var
Separator : WideString);
begin
end;
```

Delphi 8 (.NET only)

```
procedure QueryContextMenu(sender: System.Object; e:
AxEXSHELLVIEWLib._IExShellViewEvents_QueryContextMenuEvent);
begin
end;
```

Powe…

```
begin event QueryContextMenu(string Items,string Separator)
end event QueryContextMenu
```

VB.NET

```
Private Sub QueryContextMenu(ByVal sender As System.Object, ByVal e As
AxEXSHELLVIEWLib._IExShellViewEvents_QueryContextMenuEvent) Handles
QueryContextMenu
End Sub
```

VB6

```
Private Sub QueryContextMenu(Items As String,Separator As String)
End Sub
```

VBA

```
Private Sub QueryContextMenu(Items As String,Separator As String)
End Sub
```

VFP

```
LPARAMETERS Items,Separator
```

Xbas…

```
PROCEDURE OnQueryContextMenu(oExShellView,Items,Separator)
RETURN
```

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

Java…

```
<SCRIPT EVENT="QueryContextMenu(Items,Separator)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function QueryContextMenu(Items,Separator)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComQueryContextMenu String llItems String llSeparator
    Forward Send OnComQueryContextMenu llItems llSeparator
End_Procedure
```

Visual Objects

```
METHOD OCX_QueryContextMenu(Items,Separator) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_QueryContextMenu(COMVariant /*string*/ _Items,COMVariant
/*string*/ _Separator)
{
}
```

XBasic

```
function QueryContextMenu as v (Items as C,Separator as C)
end function
```

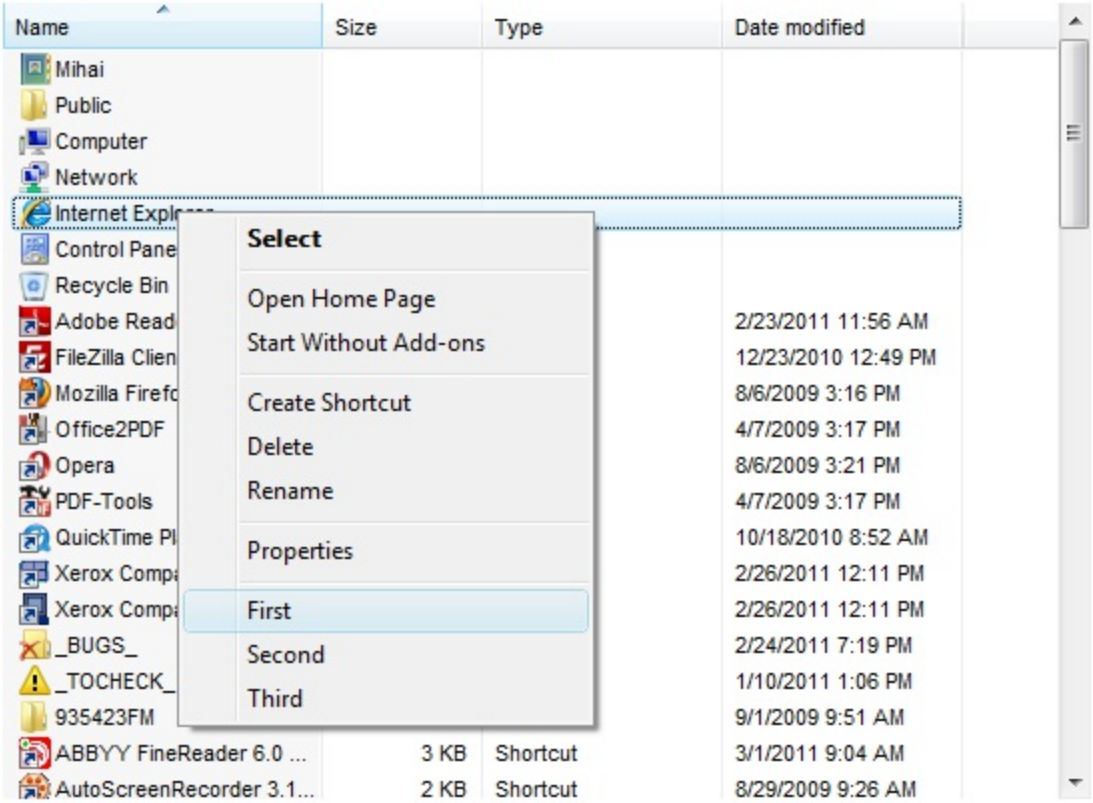
dBASE

```
function nativeObject_QueryContextMenu(Items,Separator)
return
```

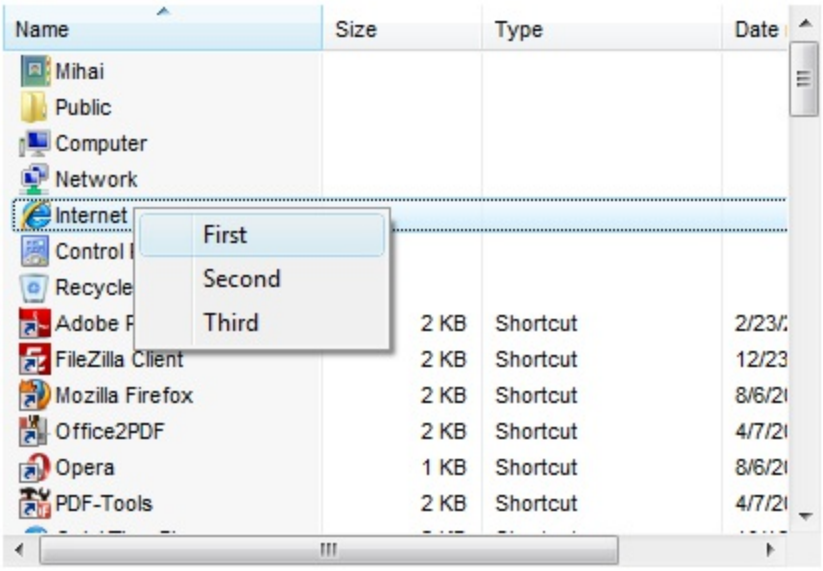
For instance the following VB sample adds 3 more items in the control's default context menu:

```
Private Sub ExShellView1_QueryContextMenu(Items As String, Separator As String)
    Separator = ","
    Items = ",First,Second,Third"
End Sub
```

and so the control's context menu shows three more items as in the following screen shot:



and if the DefaultMenuItems property is set on False, the context menu shows only the new three items:



The first separator item in the context menu is not shown because we have used *Items = "First,Second,Third"* instead *Items = ",First,Second,Third"*

event StateChange (NewState as StatesEnum)

Fired when the list's state has been changed: focus, selection.

Type	Description
NewState as StatesEnum	A StatesEnum expression that represents the current state.

This event is fired each time current state is changed. The ViewFolderFlags property may be used to enable multiple items selection.

It is fired when one of the following operation occurs:

1. When eXShellView control got the focus,
2. When eXShellView lost the focus,
3. When current selection was changed,
4. When item was renamed.

Depending on newState variable, user can make certain actions. Use the Objects property to retrieve the collection of all or selected items.

The Objects.Get method gets:

- nothing, if the objectType parameter is **NoItems**
- all files or folders being listed in the current view, if the objectType parameter is **AllItems**
- all files or folders being listed in the current view, as they are displayed, if the objectType parameter is **AllItems Or AsDisplayed**
- selected files or folders, if the objectType parameter is **SelectedItems**
- selected files or folders as they are displayed, if the objectType parameter is **SelectedItems or AsDisplayed**

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

C#

```
private void StateChange(object sender,exontrol.EXSHELLVIEWLib.StatesEnum
NewState)
{
}
```

VB

```
Private Sub StateChange(ByVal sender As System.Object,ByVal NewState As
```

```
exontrol.EXSHELLVIEWLib.StatesEnum) Handles StateChange
End Sub
```

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

C#

```
private void StateChange(object sender,
AxEXSHELLVIEWLib._IExShellViewEvents_StateChangeEvent e)
{
}
```

C++

```
void OnStateChange(long NewState)
{
}
```

C++ Builder

```
void __fastcall StateChange(TObject *Sender,Exshellviewlib_tlb::StatesEnum
NewState)
{
}
```

Delphi

```
procedure StateChange(ASender: TObject; NewState : StatesEnum);
begin
end;
```

Delphi 8 (.NET only)

```
procedure StateChange(sender: System.Object; e:
AxEXSHELLVIEWLib._IExShellViewEvents_StateChangeEvent);
begin
end;
```

Powe…

```
begin event StateChange(long NewState)
end event StateChange
```

VB.NET

```
Private Sub StateChange(ByVal sender As System.Object, ByVal e As
AxEXSHELLVIEWLib._IExShellViewEvents_StateChangeEvent) Handles StateChange
End Sub
```

VB6

```
Private Sub StateChange(ByVal NewState As EXSHELLVIEWLibCtl.StatesEnum)
End Sub
```

VBA

```
Private Sub StateChange(ByVal NewState As Long)
End Sub
```

VFP

```
LPARAMETERS NewState
```

Xbas…

```
PROCEDURE OnStateChange(oExShellView,NewState)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComStateChange OLEStatesEnum llNewState
   Forward Send OnComStateChange llNewState
End_Procedure
```

Visual Objects

```
METHOD OCX_StateChange(NewState) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_StateChange(int _NewState)
{
}
```

XBasic

```
function StateChange as v (NewState as OLE::Exontrol.ShellView.1::StatesEnum)
end function
```

dBASE

```
function nativeObject_StateChange(NewState)
return
```

The following VB6 sample gets a collection of selected items (in case your control allows multiple selection):

```
Private Sub ExShellView1_StateChange(ByVal newState As EXSHELLVIEWLibCtl.StatesEnum)
   If (newState = SelChangeState) Then
      ExShellView1.Objects.Get SelectedItems
      With ExShellView1.Objects
         For i = 0 To .Count - 1
            Debug.Print .Item(i).Name
         Next
      End With
   End If
End Sub
```

In case your control supports single selection, you can use the ObjectSelected event to notify when a new item/object is selected:

```
Private Sub ExShellView1_ObjectSelected(ByVal Object As
EXSHELLVIEWLibCtl.IExShellObject)
   If Not (Object Is Nothing) Then
      Debug.Print Object.Name
   End If
End Sub
```

The following VB.NET sample shows how to get the selected files/folder for /NET assembly version:

```
Dim i As Long = 0, s As String = ""
With Exshellview1
   .Objects.Get(exontrol.EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems)
   With .Objects
      For i = 0 To .Count - 1
         Dim sel As exontrol.EXSHELLVIEWLib.exshellobject = .Item(i)
         ' * The sel indicates the shell object being selected *
         s = s + sel.Name + vbCrLf
      Next
   End With
End With
If s.Length > 0 Then
```

```
    MessageBox.Show(s, "Selection")
Else
    MessageBox.Show("Empty", "Selection")
End If
```

The following C# sample shows how to get the selected files/folder for /NET assembly version:

```
string s = "";
exshellview1.Objects.Get(exontrol.EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems);
for ( int i = 0; i < exshellview1.Objects.Count; i++ )
{
    exontrol.EXSHELLVIEWLib.exshellobject sel = exshellview1.Objects[i];
    // * The sel indicates the shell object being selected *
    s = s + sel.Name + "\r\n";
}
if (s.Length > 0)
    MessageBox.Show(s, "Selection");
else
    MessageBox.Show("Empty", "Selection");
```

The following VB.NET sample shows how to get the selected files/folder for /COM on Window.Forms version:

```
Dim i As Long = 0, s As String = ""
With AxExShellView1
    .Objects.Get(EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems)
    With .Objects
        For i = 0 To .Count - 1
            Dim sel As EXSHELLVIEWLib.ExShellObject = .Item(i)
            ' * The sel indicates the shell object being selected *
            s = s + sel.Name + vbCrLf
        Next
    End With
End With
If s.Length > 0 Then
    MessageBox.Show(s, "Selection")
Else
```

```
    MessageBox.Show("Empty", "Selection")
End If
```

The following C# sample shows how to get the selected files/folder for /COM on Window.Forms version:

```
string s = "";
axExShellView1.Objects.Get(EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems);
for (int i = 0; i < axExShellView1.Objects.Count; i++)
{
    EXSHELLVIEWLib.ExShellObject sel = axExShellView1.Objects[i];
    // * The sel indicates the shell object being selected *
    s = s + sel.Name + "\r\n";
}
if (s.Length > 0)
    MessageBox.Show(s, "Selection");
else
    MessageBox.Show("Empty", "Selection");
```

The following VB6 sample shows how to get the selected files/folder for /COM version:

```
Dim i As Long, s As String
s = ""
With ExShellView1
    .Objects.Get (EXSHELLVIEWLibCtl.ObjectTypeEnum.SelectedItems)
    With .Objects
        For i = 0 To .Count - 1
            Dim sel As EXSHELLVIEWLibCtl.ExShellObject
            Set sel = .Item(i)
            ' * The sel indicates the shell object being selected *
            s = s + sel.Name + vbCrLf
        Next
    End With
End With
If Len(s) > 0 Then
    MsgBox s, , "Selection"
Else
    MsgBox "Empty", , "Selection"
```

```
End If
```

The following Access sample shows how to get the selected files/folder for /COM version:

```
Dim i As Long, s As String
s = ""
With ExShellView1
   .Objects.Get (EXSHELLVIEWLib.ObjectTypeEnum.SelectedItems)
   With .Objects
      For i = 0 To .Count - 1
         Dim sel As EXSHELLVIEWLib.ExShellObject
         Set sel = .Item(i)
         ' * The sel indicates the shell object being selected *
         s = s + sel.Name + vbCrLf
      Next
   End With
End With
If Len(s) > 0 Then
   MsgBox s, , "Selection"
Else
   MsgBox "Empty", , "Selection"
End If
```

The following VPF sample shows how to get the selected files/folder for /COM version:

```
local sel
s = ""
with thisform.ExShellView1
   .Objects.Get(1)
   for i = 0 to .Objects.Count - 1
      sel = .Objects.Item(i)
      s = s + sel.Name + chr(13)+chr(10)
   next
endwith
messagebox(s)
```

The following C++ sample shows how to get the selected files/folder for /COM version:

```
CString s;
```

```
CExShellObjects objects = m_shellView.GetObjects();
objects.Get( 1 );
for ( long i = 0; i < objects.GetCount(); i++ )
{
   CExShellObject sel = objects.GetItem( COleVariant( i ) );
   s = s + sel.GetName() + _T("\r\n");

}
if ( s.GetLength() > 0 )
   MessageBox( s, _T("Selection") );
else
   MessageBox( _T("Empty"), _T("Selection") );
```