Extensis Portfolio 5.0

Visual Basic Reference

Table of Contents

Overview

The purpose of this document

This document is meant as a brief overview to the Portfolio commands available via the Automation interface in Extensis Portfolio 5.0 for Windows. The examples shown here were done in Visual Basic, but users familiar with other languages capable of using Automation should be able to extrapolate from the examples for their particular languages. This document assumes a basic level of familiarity with Visual Basic.

What this document is not

This document is meant neither as an introduction to Visual Basic nor as a tutorial for how to script Portfolio to perform specific tasks.

Plenty of information is already available on how to use Visual Basic. An excellent starting point for Visual Basic information is available on the Internet at http://msdn.microsoft.com/vbasic/. This site contains a great deal of useful information on Visual Basic as well as links to other programming-related web sites.

For more information on how to script Extensis Portfolio, refer to the additional files provided on the Extensis CD. Additional information is also available on Extensis' website at http://www.extensis.com in the Portfolio product section.

Starting Portfolio

As with any application being accessed through Automation, create a variable to hold the object which references the Portfolio.document class and then create the object using the set command

Example:

```
Dim PortObj as Portfolio.document
Set PortObj = New Portfolio.Document
```

If Portfolio is already running, the VB application will use that instance of the application. Portfolio is a single-instance application, so there is no way to force a new instance of the application from within VB.

Note: Throughout much of this document, the examples will refer to an object named 'PortObj'. This is the object created above, and represents a Portfolio.Document object.

Working with Catalogs

Opening catalogs

To open a Portfolio catalog via Automation, use the `Open` function of the Document object. If the catalog has already been opened with this copy of Portfolio (either through scripting or manually), the first gallery of the catalog will come to the front and become the active gallery.

Function **Open**(*Path As String, AccessMode As Integer, Password As String*) As Integer

The acceptable values for AccessMode are as follows:

READER	= 1
EDITOR	= 2
PUBLISHER	= 3
ADMINISTRATOR	= 4

Example:
x = PortObj.Open("C:\test.fdb",4,"")

Opening a catalog on a server

To open a Portfolio catalog which is being served by a Portfolio Server, use the `OpenServer` function of the Document object. The command takes the 'address' of the catalog in the form "server/catalog", where "server" is the server's name or IP address.

Function **OpenServer**(*Path As String, AccessMode As Integer, Password As String*) As Integer

The acceptable values for AccessMode are the same as above (but be aware that served catalogs cannot be opened in Administrator mode).

Example:
x = PortObj.OpenServer("192.0.0.0/Test.fdb",3,"")

Creating catalogs

To create a new catalog, use the `new` command. By default, the catalog opens in Administrator mode.

Function **New**(*Path As String*) As Integer

Example:
x = PortObj.New("C:\test.fdb")

Working with Galleries

Once a catalog has been opened or created, the majority of commands will be directed at the Gallery object (in fact, there is no Catalog object in Portfolio's Automation model).

Selecting a Gallery

The majority of actions require a reference to a Gallery object. This tells Portfolio not only which catalog's records are to be manipulated, but which set of visible records are being worked upon. The standard use of the object is to pass in an index:

```
PortObj.Gallery(x)
```

To get the count of the open galleries in Portfolio, get the `Count` property of the Document object:

```
gCount = PortObj.Count
```

To determine the active gallery, use the `GetActive` function of the Document object. This returns the name of the active gallery. If no gallery is open when this function is called, a Visual Basic automation error will occur.

```
sActive = PortObj.GetActive
```

To convert a gallery's string name, use the `GetGalleryIndexFromName` function of the Document object.

Function **GetGalleryIndexFromName**(*GalleryName As String*) As Integer

The following example references the active gallery:

Example:

```
PortObj.Gallery(PortObj.GetGalleryIndexFromName(PortObj.GetActive))
```

Sorting a Gallery

Use the `sort` function of the Gallery object to put the records in a gallery in a particular order. Be sure to only used indexed, single-valued fields for sorting. Set the `Direction` attribute to True to sort in ascending order, False to sort in descending order.

Function **Sort**(*FieldName As String, Direction As Boolean*) As Boolean

Example:

```
x = PortObj.Gallery(1).Sort("Filename",True)
```

Adding files to a gallery

To add source files to a gallery, use the `catalog` function of the Gallery object. The `catalog` command observes all the settings set in the Cataloging Options dialog in the Portfolio application.

Function **Catalog**(*Path As String, IncludeDirs As Boolean*) As Boolean

Example:
x = PortObj.Gallery(1).Catalog("C:\Images\", True)

Any path can be used as the string; this makes it very simple to catalog folders or entire volumes, as well as individual files. To catalog subfolders of a folder or volume, set the IncludeDirs property to True.

Cataloging Options

Cataloging Options can also be controlled through the Automation interface by accessing the Options object. The Options object contains 15 properties; these properties correspond to the controls available in Portfolio's Cataloging Options dialog on the General and Rules tabs. The available properties are:

AddExtractDescription
AddExtractKeywords
AddExtractThumbnail
AddSkipFiles
ModifyMethod
- 0 – Add
- 1 – Update
- 2 – Add and Update
- 3 – Add Unconditionally
- 4 – Update Unconditionally

ParseKeywords
PathKeywords
- 0 – None
- 1 – File Name
- 2 – File and Folder Name
- 3 – Path Name
- 4 – Path Name and Volume

ThumbnailQuality
- 0 – High
- 1 – Medium
- 2 – Low

ThumbnailSize
- 0 – 112 x 112
- 1 – 256 x 256

UpdateAppendDescription
UpdateExtractDescription
UpdateExtractKeywords
UpdateExtractThumbnail
UpdateMergeKeywords
UpdateThumbnail

The following example instructs Portfolio to attempt to extract an embedded thumbnail from a source file the next time a file is cataloged in the gallery.

Example:
PortObj.Gallery(1).Options.AddExtractThumbnail = True

Working with records

Each gallery typically contains one or more record objects (though it may also contain no records). Use the record objects to iterate through a particular set of records to manipulate the values of the fields within each record. Records are typically referenced through the `AllRecords` object. This object contains a record object for each record in the selected gallery. Be aware that an index represents the current order of records within a particular gallery, so a particular record's index will change based on the contents and order of the gallery.
To access a particular record in a catalog, pass in the index for the desired record object into the `AllRecords` object.

```
PortObj.Gallery(1).AllRecords(x)
```

To get the count of all the records in a gallery, use the `Count` property.

```
rCount = PortObj.Gallery(1).AllRecords.Count
```

Note: Be aware that this returns the number of records in the gallery, which is not necessarily the same as the number of records in the entire catalog.

Working with Selections

A common use of scripts is to perform operations based on the selection of records the user has made in the catalog. To determine which records are currently selected, simply refer to the `SelectedRecords` object within a particular Gallery object. As with the `AllRecords` object, an index is used to identify a particular record object within the `SelectedRecords` object.

Example:
```
PortObj.Gallery(1).SelectedRecords(x)
```

As with the `AllRecords` object, `SelectedRecords` also has a count property. This can then be used to iterate through all the selected records in the gallery.

Selecting items

To modify the selection, use the `select and deselect` functions. The `select` function does not deselect the current selection, so it may be necessary to use the `deselect` function first to clear the selection.

Examples:
```
PortObj.Gallery(1).AllRecords(1).Select
PortObj.Gallery(1).SelectedRecords(2).Deselect
```

In addition, the Gallery object has a `SelectAll` function, which will set the selection to every record in the gallery.

Example:
```
PortObj.Gallery(1).SelectAll
```

Note: Changes to the selection may not be visible on screen until the gallery is refreshed. The selection will be accurate, but the screen may not redraw if the items being manipulated are already visible.
Note: List view does not support modifying the selection. The SelectedRecords object is accessible, but the Select and Deselect functions will not work.

Working with Fields

Each Record object contains a number of Field objects (one for each system field and one for each custom field. A particular field is identified by passing in the field's name.

```
PortObj.Gallery(1).AllRecords(1).Field(x)
```

To determine a value for a field, use the Field object's `Value` property.

```
SDesc = PortObj.Gallery(1).AllRecords(1).Field("Description").Value
```

To determine the number of fields in a catalog, get the `Count` property of a particular record.

```
fldCount = PortGal.AllRecords(1).Count
```

Determining the fields for a catalog

Because Portfolio allows the user to customize the catalog by creating their own fields, it is often useful to determine which fields are present in a particular catalog. To do this, simply iterate through the list of all the field names for a particular record (any record within a catalog will return the same results) and examine the `Name` property.

Example:

```
fldCount = PortGal.AllRecords(1).Count
For i = 1 To fldCount
   If PortObj.Gallery(1).AllRecords(1).Fields(i).Name = "FieldName" Then
            ' Found the field titled FieldName
   End If
Next
```

Changing Field Values

One of the most powerful aspects of the Portfolio scripting interface is the ability to not only read all of the fields in a record, but also to modify all of the fields (including the system fields). *Be aware that while this is a very useful tool, it is also possible to ruin a catalog by incorrectly modifying data that Portfolio uses to manage source files*. For example, incorrectly modifying the path of each record in a catalog could result in Portfolio being unable to find any records again.
Setting the field value is done in the same manner as getting the field values.

Example:

```
PortObj.Gallery(1).AllRecords(1).Field("Description").Value = "This is the new description"
```

Notes on Setting Field Values

Data Types

For scripting purposes, all values for setting field values should be passed to Portfolio as strings, regardless of the data type of the field within Portfolio. Passing the value as anything else will result in an error. The reason for this is that Portfolio uses its own internal validation routines to determine whether the data being passed in is appropriate for the field type. The following example shows the "Last Updated" field (a date field) being set by passing in a string.

Example:

```
PortObj.Gallery(1).AllRecords(1).Field("Last Updated") =  "July 4, 1776"
```

Multi-valued Fields

Field objects contain a set of functions for handling multi-valued data. For reading the multi-valued list, use the `MVDataCount` property to determine the number of items in the list, and then iterate the list using the `GetMVData` function to read the values.

Example:

```
iCount = PortObj.Gallery(1).AllRecords(1).Field("Keywords").MVDataCount
For i = 1 To iCount
   sValue = PortObj.Gallery(1).AllRecords(1).Field("Keywords").GetMVData(i)
   ' Do something with the data
Next i
```

Set the `Value` property to add a value to the multi-valued list. Use the `DeleteMVData` function to remove a particular value, or use the `DeleteData` function to remove all values from the list.

Examples:

```
'Adds Dog to the keyword list
PortObj.Gallery(1).AllRecords(1).Field("Keywords").Value = "dog"

'Removes Dog from the keyword list
result = PortObj.Gallery(1).AllRecords(1).Field("Keywords").DeleteMVData("dog")

'Removes all the keywords from this item
result = PortObj.Gallery(1).AllRecords(1).Field("Keywords").DeleteData
```

Searching the catalog

To perform a search in a Portfolio catalog, use the `find` function in the Gallery object. Searches in Portfolio are executed by passing in a text string which represents the search criteria as they are laid out in the Find dialog in Portfolio. The basic functionality is shown below. See the next section on how to formulate the query variable properly.

Function **Find**(*SearchString As String, AllRecords As Boolean, SetNewGallery As Boolean*) As Integer

The `AllRecords` attribute is the equivalent of the "Find in Gallery" checkbox in the Portfolio Find dialog. A value of True is the equivalent of that box being unchecked. The `SetNewGallery` attribute is the equivalent of the "Display Results in New Gallery" checkbox in the Portfolio Find dialog. A value of True is the equivalent of that box being checked.

To find all the records in the catalog, simply pass in a query that cannot fail.

Example:
PortObj.Gallery(1).Find(("Filename" & vbTab & "starts with" & vbTab & ""), True, False)

Building the Query

Searches in Portfolio are executed by passing in a text string which represents the search criteria as they are laid out in the Find dialog in Portfolio. As in the Portfolio Find dialog, the basic query structure consists of three clauses: the field, the operator, and the value. Each of these clauses is passed in as a textual string, and the tab character separates each clause.

Example:
theQuery **=** "Keywords" & vbTab & "starts with" & vbTab & "test"
PortObj.Gallery(1).Find(theQuery, True, False)

To build a more complex search, a return character must be used to delimit each line. In addition, each line after the first one needs to begin with the join condition (either "and" or "or"). Below is an example of a two line search query.

Example:
theQuery = "Filename" & vbTab & "starts with" & vbTab & "test" & vbNewLine & "and" & vbTab &
"Keywords" & vbTab & "starts with" & vbTab & "key"

Importing and Exporting

The Portfolio scripting interface also provides for importing and exporting data.

Exporting HTML

The scripting interface for the Export HTML command is very similar to the Export HTML command within Portfolio. To export the current selection using the layout of the gallery as the template, use the `ExportHTML` function of the Gallery object. As in Portfolio, the Export HTML command is selection-based, so a record set must be selected for the command to work correctly.

> Function **ExportHTML**(*HTMLPath As String, HTMLName As String, SavedSetName As String*) As Integer

The `HTMLPath` attribute is a string pointing at the destination directory. The `HTMLName` attribute is a string with the name of the first HTML file to be generated. The `SavedSetName` is the name of the HTML Template to be used. To use the current gallery's settings, simply pass in an empty string.

> **Examples:**
> result = PortObj.Gallery(1).ExportHTML("C:\Test\", "myHTML.htm", "")
> result = PortObj.Gallery(1).ExportHTML("C:\Test\", "myHTML.htm", "MySavedTemplate")

Notes: Any alerts that might come up during the Export process (such as overwrite warnings) are automatically overridden. If files of the same name already exist at the location specified, they will be automatically overwritten.
The template specified must already exist in the selected catalog; it cannot be created through the scripting interface.

Importing text data

To import text data using Portfolio's Import Field Values command, use the `ImportFieldValues` function of the Gallery object. The path passed in is the text file to be used, and the `SavedSet` property specifies the Import Saved Set to be used for matching up the incoming data with Portfolio fields.

Function ImportFieldValues(Path As String, SavedSet As String) As Boolean

> **Example:**
> result = PortObj.Gallery(1).ImportFieldValues("C:\Test\TestFile.txt", "MySavedSet")

Note: The template specified must already exist in the selected catalog; it cannot be created through the scripting interface.

The Portfolio_V5 Interface

All objects, properties, and methods of the Portfolio interface (as described in the rest of this document) are supported in the Portfolio_V5 interface. In addition, the Portfolio_V5 interface has two functions: "OpenByUserName" and "OpenServerByUserName" to allow for the opening of catalogs with user-based access security. To access the interface, create an object variable to hold a reference to the Portfolio_V5.Document class and create the reference using the set command.

Example:

```
Dim Port5Obj as Portfolio_V5.Document
Set Port5Obj = New Portfolio_V5.Document
```

Opening catalogs with User-based access

To open a catalog with user-based access security, use the Document.OpenByUserName function for catalogs on disk, or the Document.OpenServerByUserName function for served catalogs (native or via the SQL service). The catalog is automatically opened at the maximum access mode allowed for the specified user.

```
Function OpenByUserName (Path As String, UserName As String, Password As String) As Integer
Function OpenServerByUserName (Path As String, UserName As String, Password As String) As Integer
```

Example:

```
x = Port5Obj.OpenByUserName("C:\test.fdb","User1", "password")
x = PortObj.OpenServerByUserName("192.0.0.0/Test.fdb","User1","password")
```