Creative assets management

Extensis™

Portfolio 8™

Visual Basic Guide

Contact

Extensis

1800 SW First Avenue,
Suite 500
Portland, OR 97201
Toll Free: (800) 796-9798
Phone: (503) 274-2020
Fax: (503) 274-0530
http://www.extensis.com

Extensis Europe

First Floor, Century House
The Lakes
Northampton NN4 7SJ
United Kingdom
Phone: +44(0)1604 636 300
Fax +44 (0)1604 636 366
info@extensis.co.uk

Celartem, Inc.

Email: sales_ap@celartem.com
http://www.celartem.com/jp/

Press Contact

Phone: (503) 274-2020 x129
Email: press@extensis.com

Customer Service

Web/email: *http://www.extensis.com/customerservice/*
Phone: (800) 796-9798

Technical Support

Web/email: *http://www.extensis.com/support/*

Documentation Feedback

Web/email: *http://www.extensis.com/helpfeedback/*

Contents

Visual Basic and Portfolio

This document provides a brief overview of the Portfolio commands available via the Automation interface in Extensis Portfolio 8.0 for Windows. The examples covered here are written in Microsoft Visual Basic 6. The source code for the sample applications is provided in both Visual Basic 6 and Visual Basic .NET (2003) format. 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 working familiarity with Visual Basic.

This document is not meant to be an introduction to Visual Basic or as a tutorial for how to script Portfolio to perform specific tasks. For more information about the basics of Visual Basic, please visit *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 websites. For more information on how to script Extensis Portfolio, please refer to the additional files provided on the Extensis CD, or with your product download.

For more information about Portfolio, please refer to the Portfolio User Guide installed with your Portfolio client. For the most up-to-date information about Portfolio, please visit the Extensis website at *http://www.extensis.com/portfolio/*.

Introduction to the Portfolio Automation Interface

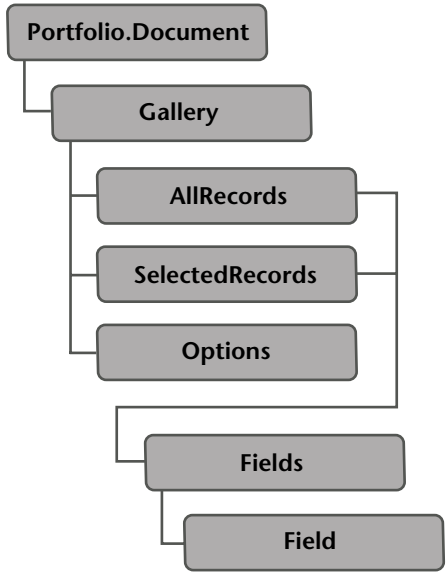
How to add Portfolio to your Visual Basic project

All objects, properties and methods of the Portfolio interface (as described in the rest of this document) are supported in the Portfolio_V8 automation interface.

To expose the Portfolio object, and its family of classes, to your Visual Basic code, you must first include it in the reference options in your VB project. Use the Project References dialog within the VB interface to select the Portfolio_V8 library. Once this has been done, your VB application will be aware of the Portfolio document interface and you can reference the family of classes within your VB code.

The Portfolio Object Model:

The diagram below shows the relative structure of the Classes within the **Portfolio Library Object Model**.



One thing you'll notice in the Portfolio Object Model is that there is no Catalog object or class. The Portfolio document object is actually a collection of Galleries. If you need to reference the Catalog name, you can use the `Gallery.Catalog` name property.

Creating Portfolio Objects

As with any application being accessed through Automation, you need to create a variable to hold the object which references the `Portfolio _ V8.document` class and then create the object using the set command

Example:

```
Dim PortObj as Portfolio _ V8.document

Set PortObj = New Portfolio _ V8.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.

To access the Gallery class, create an object variable to hold a reference to the `Portfolio _ V8.Gallery` class, and then create the reference using the set command.

Example:

```
Dim PortGal as Portfolio _ V8.Gallery

Set PortGal = New Portfolio _ V8.Gallery
```

Throughout much of this document, the examples will refer to objects named `PortObj` and `PortGal`. These are the objects described above, and represent `Portfolio _ V8.Document` and `Portfolio _ V8.Gallery` objects. These objects are the communication interfaces that allow your VB code to access the features and functionality of Portfolio.

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,"")
```

Working with Catalogs

Opening Catalogs on a Server

To open a 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. Be aware, however, that server catalogs can't be opened via your VB code in the Administrator mode.

Example:

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

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 = PortObj.OpenByUserName("C:\test.
fdb","User1", "password")

x = PortObj.OpenServerByUserName("192.0.0
.0/Test.fdb","User1","password")
```

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, most of the functionality of the Portfolio document model is managed through the Gallery object. As noted elsewhere, there isn't even a catalog object available.

Selecting a Gallery

The majority of actions in your code will be based upon a reference to a `Gallery` object. This reference tells Portfolio not only which catalog's records are to be manipulated, but which set of visible records (images) are being targeted. Defining which gallery is targeted with your code requires referencing an index with the gallery object:

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

To obtain the number of open galleries in the Portfolio document, use 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. See Appendix D for a full listing of error codes.

```
sActive = PortObj.GetActive
```

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

```
Function GetGalleryIndexFromName(Gallery
Name As String) As Integer
```

The following example references the active gallery:

Example:

```
PortObj.Gallery(PortObj.GetGalleryIndexFr
omName(PortObj.GetActive))
```

Sorting a Gallery

You can use the sort function of the `Gallery` object to organize the records in the active gallery in a particular order. Be sure to only use indexed, single-valued fields for sorting. To determine the sort order, set the `Direction` attribute to `True` to sort in ascending order and `False` to sort in descending order.

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

Example:

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

Saving A Gallery

To save an existing gallery that has been changed, use the `Gallery.Save` function for a gallery that already exists, or use the `Gallery.SaveAs` function for saving new galleries.

```
Function Save() As Integer

Function SaveAs(sNewName As String) As
Integer
```

Example:

```
x = PortGal.Save()

x = PortGal.SaveAs("gallery")
```

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 found 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 - Advanced Options

How to access the Advanced Options

Cataloging options can also be controlled within your Visual Basic application using the Portfolio Automation interface. These are accessed via the AdvancedOptions class of the Portfolio object. The AdvancedOptions object contains properties that correspond to the controls available in the **Cataloging Options** and **Advanced Options** dialog boxes. These are found in the Portfolio interface in the Catalog menu.

Catalog Property Listing

The available properties of the class are:

```
AppendDescription
ExtractMetaData
ExtractThumbnail
SkipFiles
MergeKeywords
ParseKeywords
PathKeywords
   0 - None
   1 - File Name
   2 - File and Folder Name
   3 - Path Name
   4 - Path Name and Volume
ThumbnailSize
   0 - 112 x 112
   1 - 256 x 256
```

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).AdvancedOptions.
ExtractThumbnail = True
```

Working with Records & Selections

Accessing Individual Records

Each gallery typically contains one or more record objects. If, however, it is a new gallery, it may contain no records. Use the records properties of the gallery class to iterate through a 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 number 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
```

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

One common use of scripts (or Visual Basic programs) 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 that will return the numbers of individual records selected. This can then be used to define a loop in your Visual Basic code and step 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 point to include every record in the gallery.

Example:

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

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.

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

Portfolio's dynamic data structure allows the user to customize the catalog by defining custom fields. Since you might be working with a record structure that contains custom data fields, it is often useful to determine which fields are defined in a particular catalog. To do this, step 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

      rem 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"
```

You can also find an example of Visual Basic code that performs a search and replay function on fields in the Portfolio tables in the **Path Change** utility application that accompanies this document.

Proceed with caution when you are learning to manipulate the content of fields within the Portfolio data tables. It is possible to make data changes that could corrupt your entire catalog. Always back up your Portfolio data before trying to make changes to the data within the tables from within your Visual Basic code.

Passing Field Values to Portfolio

Data Types

When programming with the Portfolio objects, all values used to set field values should be passed to Portfolio as strings, regardless of the data type of the field within Portfolio. Passing the value as anything other than a string variable or value will result in an error. Portfolio uses its own internal validation routines to determine whether the data being passed in is appropriate for the field type, and that internal process will only accept incoming string variable data. The following example shows the `Last Updated` field (a date field) being set by passing a string value.

Keep this in mind if your application is going to be using a declared variable. If you have a date variable, for example, you must convert it to a string equivalent before passing the value to Portfolio.

Example:

```
PortObj.Gallery(1).AllRecords(1).
Field("Last Updated").value = "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)

   rem 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:

```
rem Adds Dog to the keyword list

PortObj.Gallery(1).AllRecords(1).
Field("Keywords").Value = "dog"

rem Removes Dog from the keyword list

result = PortObj.Gallery(1).AllRecords(1).
Field("Keywords").DeleteMVData("dog")

rem Removes all the keywords from this
item

result = PortObj.Gallery(1).AllRecords(1).
Field("Keywords").DeleteData
```

Searching the Catalog

How to Search

To perform a search in a Portfolio catalog, use the find function of the `Gallery` object. Searches in Portfolio are executed by passing in a text string which represents the search criteria. The format of the search string should follow the format used in the **Find** dialog in Portfolio. The basic functionality is shown below. See the next section for 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** option of the Portfolio Find dialog box. 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** option in the Portfolio Find dialog box. 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 a text string to the `Document` object which represents the search criteria. The structure of this query information is the same as it is presented in the **Find** dialog box within Portfolio. As in the Portfolio **Find** dialog box, the basic query structure consists of three clauses: the `field`, the `operator`, and the `value`. Each of these clauses is passed in as a text 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”
```

Appendix A: The Portfolio Class Libraries

This appendix contains an alphabetical listing of all of the members, functions and properties of each of the Portfolio classes exposed through the automation model.

The Portfolio Document Class:

Count as Long

Gallery(nIndex as Long) as Object [read-only]

GetActive() as String

GetGalleryIndexFromName(GalleryName as String) as Integer

LastError as Long

New(Path as String) as Integer

NewCatalogWithPrompt() as Integer

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

OpenByUserName(Path As String, AccessMode As Integer, UserName As String, Password As String) As Integer

OpenServer(sAddress As String, AccessMode As Integer, Password As String) As Integer

OpenServerByUserName(sAddress As String, AccessMode As Integer, UserName As String, Password As String) As Integer

OpenServerWithPrompt(sAddress As String) As Integer

OpenWithPrompt(sPath As String) As Integer

SetActive(GalleryName As String) As Integer

Visible as Boolean

The Portfolio Gallery Class:

Function **Activate**() As Integer

Function **AddFiles**(pDataObj As Unknown) As Boolean

Function **AddRecord**(szFile As String, szJPEGThumbnailFile As String, nThumbSize As Integer, nThumbQuality As Integer) As Object

AllRecords As Object

Busy as Long

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

CatalogName as String

Function **CatalogUsingOptionsPreset**(Path As String, szPresetName As String, IncludeDirs As Boolean) As Boolean

Function **Close**() As Integer

Function **CollectFiles**(szFileCollectFolder As String, bKeepHierarchy As Boolean, bCollectOriginals As Boolean, nPreviewSizeInPixels As Integer, bOrigIfNoPreview As Boolean, bCreateArchiveCatalog As Boolean, szCatalogPath As String, szVolumeName As String, bOverwriteExistingCatalog As Boolean, bIncludeBrowser As Boolean, bIncludeUserGuide As Boolean) As Integer

Count as Long

Function **CreateCustomField**(szFieldName As String, nFieldType As Integer, bMultiValued As Boolean, nFieldOption As Integer, szAdminPassword As String) As Boolean

Function **CreateCustomFieldPreDef**(szFieldName As String, nFieldType As Integer, bMultiValued As Boolean, nFieldOption As Integer, bFromListOnly As Boolean, szFieldValueList As String, szAdminPassword As String) As Boolean

Function **CreatePlaceholder**(szFileName As String, szOptionalPath As String) As Long

Function **DeleteRecord**(RID As Long, bDeleteFromCatalog As Boolean) As Boolean

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

GalleryName as String

Function **GetRecordFromRecordID**(nRecordID As Long) As Object

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

NumFound as Long

Options as Object

Sub **RefreshView**()

Function **Save**() As Integer

Function **SaveAs**(sNewName As String) As Integer

Function **SelectAll**() As Integer

SelectedRecords As Object

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

TotalRecordsInCatalog As Long

The Portfolio Record Class:

Count as Long

Function **Delete**(bDeleteFromCatalog As Boolean) As Boolean

Deselect() as Boolean

Sub **EnsureVisible**()

Function **ExtractProperties**() as Boolean

Property **Field**(FieldName as String) as Object

Property **Fields**(nIndex as Long) as Object

Function **GetThumbnailData**(pData, nSize As Long) As Boolean

IsSelected as Boolean

RecordID as Long

Function **RegenerateThumbnail**() As Integer

Function **Select**() As Boolean

Function **Update**() As Boolean

The Portfolio Field Class:

DeleteData() as Boolean

DeleteMVData(Value As String) As Boolean

GetMVData(nIndex As Long) As String

GetPredefinedValue(Index As Integer) As String

GetPredefinedValueCount() As Integer

GetPredefinedValueList() as String

IsCustom as Boolean

IsIndexed as Boolean

IsMultiValue as Boolean

IsPredefined as Boolean

IsURL as Boolean

MyDataCount as Long

Name as String

Type as Integer

Value as String

The Portfolio Records Class:

Count as Long

Function **GalleryCopy**(nGalleryIndex as Long)as Integer

Property **Records**(nIndex as Long) as Object [read-only]

The Portfolio Selection Class:

Count as Long

Function **GalleryCopy**(nGalleryIndex as Long) as Integer

Property **Records**(nIndex as Long) As Object [read-only]

The Portfolio AdvancedOptions Class:

AppendDescription as Integer

ExtractMetadata as Boolean

ExtractThumbnail as Boolean

MergeKeywords as Integer

ParseKeywords as Integer

PathKeywords as Integer

SkipFiles as Boolean

ThumbnailSize as Integer

APPENDIX B: Frequent Questions and Answers

This appendix contains some of the most frequently asked questions about VB programming with the Portfolio document object.

How do I add the Portfolio references to my Visual Basic Project?

From the **Project** menu, select the references option. Find the Portfolio_V8 library reference, select it and click the **OK** button. The Portfolio document object will now be included in your VB source project and its objects and properties exposed for use in your VB code. See the beginning of the tutorial for examples of the code required to declare and set a Portfolio document object within your project.

Can I use VBA or VB Script with Portfolio, or do I have to use the full version of Visual Basic?

There are major differences in the way objects are defined and handled in the full version of Visual Basic and in VBA and VB Scripting. Due to these differences, Portfolio only supports the use of the full version of Visual Basic or Visual Basic.net.

What is the PortObj variable you keep referencing?

In all of the sample code, and this document, we've used the `PortObj` variable name to identify the Portfolio document object being referenced in the code. You are free to use any variable name you choose in your own code, but for purposes of clarity, we use the standard object name throughout the documentation and sample code.

Can I use the Portfolio automation library with VB.Net?

The automation object model use by Portfolio will work with VB.net, but you need to keep in mind that if you develop any applications using the Portfolio automation object in VB.net, the Net Framework will have to be installed on any machine that will be running your application.

Where can I find the field definitions for the Portfolio tables?

While in Portfolio, use the **Customize View** option to see a full listing of fields included in a Catalog. You can use this to select which fields you wish to see displayed. Use this as a debugging tool when you are writing code to edit or change field values. It provides instant feedback to see if you code executed the changes you expected in the field values.

If you wish to read or display the list of available fields from within your Visual Basic application, see the **Working with Fields** section of the tutorial.

How can I tell how many Galleries are open?

You can get a count of the current open Galleries with the following code:

```
PortObj.Document.Count
```

You need to be aware that this Count value will return the number of all Galleries that are currently open in all open Catalogs. For example, if you have two open Catalogs, with three Galleries each, and one Gallery is open in each Catalog, the Count value you receive will be two.

Can I activate a particular Gallery from my VB code?

Yes. Use the following command:

```
PortObj.SetActive("CatalogName.fdb -
GalleryName")
```

Be aware that you must use the full Gallery name, including the parent catalog. You can't just use the name of the Gallery itself. You can find the full Gallery name by looking at the Gallery window's caption bar in Portfolio.

How can I get the Catalog name?

As you've probably noticed, with no Catalog class or object there isn't an obvious way to read the Catalog name. Remember that the document object is actually a collection of Galleries, and a `Gallery` object has a `Catalog Name` property. You can use this to read the name of the current catalog:

```
NameVariable = PortGal.CatalogName
```

You can also reference the Catalog name by looking at the Gallery.Name property which will include the Catalog name as part of the Gallery name definition.

For example:

```
Gallery Name: ("MasterCatalog.fdb -
Sample Gallery")
```

Why do I keep getting errors when trying to set field values?

See the section on **Passing Field Values to Portfolio** in the tutorial. The most common cause of this error will be that you are passing non-string values to the Portfolio object. It expects all field values, regardless of the type, to be passed as string values.

APPENDIX C:
Guide to The Sample Applications

Portfolio Automation Demo:

While this application provides a variety of utility functions, its greatest value to the VB programmer lies in the wide range of Portfolio class objects and methods it uses. By stepping through the code as it runs, you can follow many of the key programming functions available to you as you learn to program using the Portfolio document object and its classes.

Path Change Utility:

This is a helpful utility that will let you edit the path definitions within the Portfolio database if you have moved cataloged images from one location to another.

You should be aware that there are two fields that hold the path information for a record (`Path` and `Directory Path`) and if you wish to redefine the path info for a record you must change both of these fields.

You can also examine the code to find an example of how to read and write field values from within your VB code.

Backup Utility:

This is a simple utility program that will let you back up an existing Portfolio catalog. While it doesn't use any of the Portfolio automation methods, it does provide an example of how to stop and start the Portfolio server from within your VB code using predefined `.BAT` files.

APPENDIX D: Portfolio Programming Error Codes

This appendix contains an alphabetical listing of all of the members of each Portfolio class object.

When an error occurs, the Portfolio error code value can be found by reading the `ObjectName.LastError` property. Any time a function or call succeeds that error value is set to zero.

Use standard Visual Basic error-trapping procedures to test and handle these errors. (See your VB reference books if you aren't sure how to manage error handling within your VB application.)

KScriptNoError	**0**	**No Error (Success)**
KScriptBadPassword	**1**	**Incorrect/Invalid Password**
KScriptCatInUse	**2**	**Catalog is currently in use and therefore cannot be opened in this mode. This error code is not actually used in the Portfolio client code and therefore should not be encountered.**
KScriptAdminCatInUse	**3**	**Attempting to open a Catalog in Admin mode whilst there are other users**
KScriptAdminInUse	**4**	**Someone else is already logged in as Admin**
KScriptReadOnlyAvailable	**5**	**Catalog is only available in read-only mode, for example on a read-only volume**
KScriptServedAdminError	**6**	**Opening a served Catalog in Admin (or Browser) mode**
KScriptUnknownCatError	**7**	**Unknown error trying to open Catalog**
KScriptInvalidMode	**8**	**Invalid mode**
KScriptNoFileName	**9**	**No Filename**
KScriptDiskFull	**10**	**Disk full**
KScriptErrorCreate	**11**	**Error creating Catalog**
kScriptInvalidFieldname	**12**	**Invalid Field name**
kScriptFieldNotIndex	**13**	**Field isn't indexed**
kScriptFieldNotMultiVal	**14**	**Field isn't multivalue**
kScriptIndexOutOfRange	**15**	**Index out of range**

kScriptEmptyValue	16	Empty value
kScriptInvalidValueforFieldType	17	Value is invalid for given Field type
kScriptValueNotInPredefList	18	Value is not in pre-defined list
kScriptCatalogNotFound	19	Catalog could not be found
kScriptServerNotFound	20	Server could not be found
kScriptMaxServerConnected	21	Maximum allowed number server connections reached
kScriptInvalidFieldValue1	22	Invalid FieldValue Field 1
kScriptInvalidFieldValue2	23	Invalid FieldValue Field 2
kScriptInvalidGallery	24	Invalid Gallery
kScriptInvalidQuery	25	Invalid Query (Find) specification
kScriptSystemField	26	Can't change system (built-in) Field
kScriptGalleryNotFound	27	Gallery Not Found
kScriptFieldNotPredefined	28	Field Not of pre-defined Type
kScriptInvalidCatalogOption	29	Invalid Cataloging option
kScriptInvalidHTMLSetName	30	Invalid HTML saved set name
kScriptCreatingHTMLImagesDir	31	Error Creating HTML Images Directory
kScriptCreatingHTMLFile	32	Error Creating HTML File
kScriptWritingHTMLFile	33	Error writing to HTML file
kScriptOutOfMemory	34	Out of memory
kScriptCopyError	35	Copy error (GalleryCopy)
kScriptServerDisconnect	36	Server disconnected
kScriptInvalidForFoundItems	37	Can't perform on "found items" gallery
kScriptInvalidLoginMode	38	Attempting to open user-based catalog with level-based API
kScriptUnknownError	1000	Unknown Error opening/creating catalog

P8_VBG_0506