Manage Strings Easily in VB6

Build a powerful utility class by combining the new Split and InstrRev functions with the FileSystemObject.

by Chris Barlow and Stan Schultes

Many people think computer programmers spend their days writing complex mathematical algorithms. I've lost track of the number of times folks have told me they'd like to be programmers, but weren't very good in math. In reality, most programmers spend little time writing math functions. Instead, they write functions to manipulate strings.

For example, programmers break apart name and address data blocks into first name, last name, street, city, and ZIP code to write them into a database, or they read these fields from a database and join them together in a report. They break apart a comma-delimited string into separate database fields, or they join database fields together to create a delimited export file. With so much breaking apart and joining together, programmers need good tools. At the top of the heap is Visual Basic 6, which has some neat new functions that make it easy to manipulate strings (see Table 1).

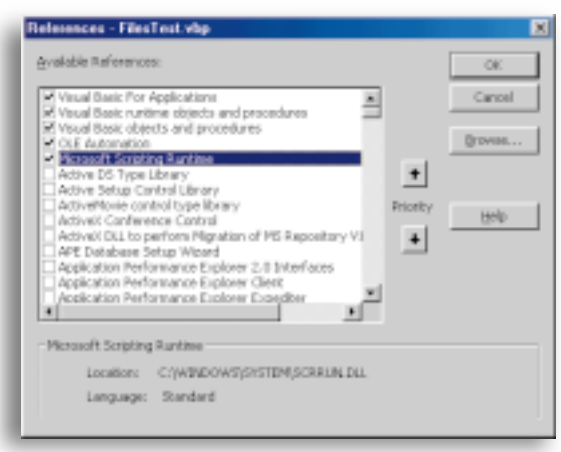

Figure 1 Set the Project References. When you set your project's references, you instruct Visual Basic to load the type libraries for these applications and early-bind your application's objects to the specific classes. You need to reference the Microsoft Scripting Runtime library to use the FileSystemObject and File classes.

Here's an example: Suppose you have a fully qualified path and file name, such as \\chrisb\c-drive\vb\apps\vbpj\1998\9810\filestring.exe, and you want to extract only the file name from the string. Before VB6, you needed to write a procedure that started at the back of the string and used a descending For…Next loop, with a Step of –1, to look for the last backslash in the string:

```
Function StripPathName(fname As _
   String) As String
'returns file name
Dim i As Integer
For i = Len(fname) To 1 Step -1
   If Mid(fname, i, 1) = _
      "\" Then Exit For
Next
StripPathName = Trim$(Mid$(fname, i + _
   1))
End Function
```

With the new InstrRev function, which returns the position of the last occurrence of one string within another, you can do the same thing with one line of code:

```
Public Function StripPathName(fname As _
   String) As String
StripPathName = Trim$(Mid$( fname, _
   InStrRev(fname, "\")))
End Function
```

The new Split function lets you parse a delimited string into an array with a single statement. It demonstrates another useful addition in VB6—functions can return an array. You can specify the delimiter used to parse the string and the maximum number of substrings you want returned. For example, you can split all the elements of a fully qualified path and file name into a single one-dimensional array with this line of code:

```
ParsePath = Split(sPathAndFileName, "\")
```

Build a File String Class

These new functions are so useful to manipulate paths and file names that it makes sense to wrap them into a class you can use in all your applications. If you combine this class with the new VB6 FileSystemObject, which gives several properties and methods to access the file system, you can build a powerful utility class. The VB6 FileSystemObject is built into the Microsoft Scripting Runtime library.

Start VB6 and create an ActiveX EXE project. Select the References menu item from the Project menu, and select the Microsoft Scripting Runtime library contained in SCRRUN.dll (see Figure 1). Insert a new class and name it CFileString. The class has two private variables: one to store the fully qualified path and file name, and one to reference the FileSystemObject (download Listing 1 from the free, Registered Level of DevX; see the Code Online box for details):

```
Private m_sPathAndFileName As String
Private m_fso As New FileSystemObject
```

Although the FileSystemObject is only used within the class, the fully qualified path and file name are exposed through public Property Get and Let procedures:

```
Public Property Get PathAndFileName() _
   As String
   PathAndFileName = m_sPathAndFileName
End Property

Public Property Let PathAndFileName( _
   ByVal NewPathAndFileName As String)
```

Function	Returns	Syntax
Filter	A zero-based array containing a subset of a string array based on a specified filter criteria	Filter(InputStrings, Value[, Include[, Compare]])
FormatCurrency	An expression formatted as a currency value using the currency symbol defined in the system control panel	FormatCurrency(Expression[,NumDigitsAfterDecimal [,IncludeLeadingDigit [,UseParensForNegativeNumbers [,GroupDigits]]]])
FormatDateTime	An expression formatted as a date or time	FormatDateTime(Date[,NamedFormat])
FormatNumber	An expression formatted as a number	FormatNumber(Expression[,NumDigitsAfterDecimal [,IncludeLeadingDigit [,UseParensForNegativeNumbers [,GroupDigits]]]])
FormatPercent	An expression formatted as a percentage (multiplied by 100) with a trailing % character	FormatPercent(Expression[,NumDigitsAfterDecimal [,IncludeLeadingDigit [,UseParensForNegativeNumbers [,GroupDigits]]]])
InstrRev	A Variant (Long) specifying the position of the first occurrence of one string within another	InStr([start,]string1, string2[, compare])
Join	A string created by joining a number of substrings contained in an array	Join(list[, delimiter])
MonthName	A string indicating the specified month	MonthName(month[, abbreviate])
Replace	A string in which a specified substring has been replaced with another substring a specified number of times	Replace(expression, find, replacewith[, start[, count[, compare]]])
Round	A number rounded to a specified number of decimal places	Round(expression [,numdecimalplaces])
Split	A zero-based, one-dimensional array containing a specified number of substrings	Split(expression[, delimiter[, count[, compare]]])
StrReverse	A string in which the character order of a specified string is reversed	StrReverse(string1)
WeekdayName	A string indicating the specified day of the week	WeekdayName(weekday, abbreviate, firstdayofweek)

Table 1 New String Functions. VB6 contains 13 new string functions. Several of them operate with dates to return locale-specific month and weekday names.

```
   m_sPathAndFileName = _
      Trim$(NewPathAndFileName)
End Property
```

You can include a method to return the file name using the StripPathName procedures and add a StripFileName method that returns only the path using the new InstrRev function:

```
Public Function _
   StripFileName(Optional _
   sFile As String) As String
   Dim i As Integer
   Dim sFile as String
   On Error Resume Next
   If Len(sFile) = 0 Then sFile = _
      m_sPathAndFileName
   i = InStrRev(sFile, "\")
   StripFileName = Left$(sFile, _
      i - 1)
End Function
```

It's also easy to write a Property Get procedure to return the file's extension by using the InstrRev function to search for the last period in the string. By not including a Property Let procedure, you can make this property read-only so the user can't set the value of this property. Note that such a property is similar to having a single method that returns a value. The choice is really up to you—there's no clear distinction between these implementations. As the developer, you need to decide which procedure is easier to understand:

```
Public Property Get FileExt() _
   As String
   Dim i As Integer
   i =
      InStrRev( _
         m_sPathAndFileName, ".")
   FileExt = _
      Mid$(m_sPathAndFileName, _
      i + 1)
End Property
```

You can also easily add the ParsePath property to the class. This property uses the new Split function to return an array of all the drives and folders that make up the fully qualified path and file name. Notice how the function definition uses the new As String() declaration to show an array is being returned:

```
Public Property Get ParsePath() As _
   String()
   ParsePath = _
      Split(m_sPathAndFileName, _
      "\")
End Property
```

Another useful method, CompressPath, takes a long, fully qualified path and file name and returns a compressed version such as \\chrisb\c-drive\…\filestring.exe. This method uses the Split function to parse the path into its individual folders. It then creates a new four-element array and copies the first two elements, an ellipsis, and the last element into the new array and uses the new Join function to create a delimited string from the new array:

```
Public Function CompressPath() _
   As String
   Dim sArray() As String
   Dim sNewArray(3) As String
   Dim i As Integer
   sArray = _
      Split(m_sPathAndFileName, _
      "\")
   i = UBound(sArray)
   If i > 2 Then
      sNewArray(0) = sArray(0)
      sNewArray(1) = sArray(1)
      sNewArray(2) = "..."
      sNewArray(3) = sArray(i)
      CompressPath = _
         Join(sNewArray, "\")
   Else
      CompressPath = _
         m_sPathAndFileName
   End If
End Function
```

Using the FileSystemObject

Visual Basic 6 lets you use the FileSystemObject class to add some functionality to your class. The FileExists method of the FileSystemObject class lets you easily add a Boolean FileExists property to indicate whether the file actually exists at the specified path:

```
Public Property Get FileExists() As _
   Boolean
   FileExists = m_fso.FileExists( _
      m_sPathAndFileName)
End Property
```

Similarly, you can add a FolderExists property by combining the FileExists method of the FileSystemObject class with the StripFileName method of your class:

```
Public Property Get FolderExists() _
   As Boolean
   FolderExists = _
      m_fso.FolderExists( _
      StripFileName())
End Property
```

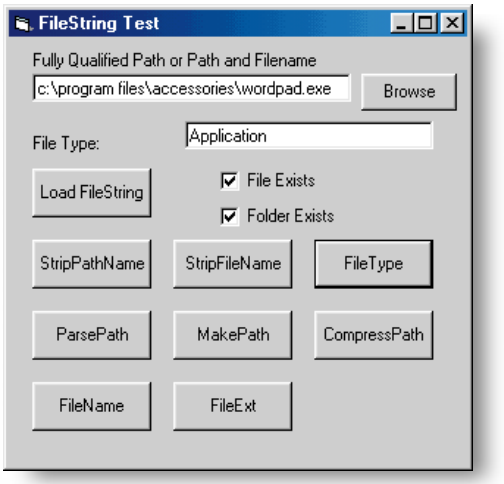

Figure 2 Create a Test Harness. The test harness lets you test your class. Notice that the Browse button uses the ShowOpen method of the Common Dialog control to display the File Open dialog and fill the text box with the fully qualified path and file name.

The FileSystemObject class can also return the type of an existing file based on its Registry setting. This useful capability is worth adding to your class. For example, a VBP file has a type of Visual Basic Project. The FileSystemObject class has a Type property, but it only works for an existing file. You first need to define a temporary variable as an instance of the Microsoft Scripting Runtime library's File class.

The GetFile method of the FileSystemObject class fills the File class. Take some time to explore the File class with the Object Browser. The File class gives you a complete set of properties for the specified file, including its type, size, creation date, and so on. It also gives you a similar set of properties for all the folders in the path, including a collection of the other files in the folder. It even has a complete set of properties for the drive, including the volume name. This procedure returns only the file type; you might want to add property procedures to return some of the file's other properties:

```
Public Property Get FileType() As _
```

```
      String
   Dim filTemp As File
   If FileExists Then
      Set filTemp = m_fso.GetFile( _
         m_sPathAndFileName)
      FileType = filTemp.Type
   End If
End Property
```

One problem when storing files on a disk is that you must create all the folders in the fully qualified path individually, beginning with the top-most folder. Adding a MakePath method to your class allows you to create all the folders in a long path with a single call. First, add the MakePath method that calls a private MakeFolder method for the complete path:

```
Public Function MakePath() As _
   Boolean
   MakePath = _
      MakeFolder(StripFileName())
End Function
```

Then write a MakeFolder procedure that can call itself recursively. That is, this procedure creates the path that is passed in as an argument using the CreateFolder method of the FileSystemObject class. It first creates the parent folder (if it doesn't already exist) by calling itself with the parent folder's path as an argument. It uses the FolderExists method of the FileSystemObject class to test whether the folder already exists. If you pass in a path with four new folders, it's fun to watch this process in single-step debug mode as it calls itself three more times to create each parent folder:

```
Private Function MakeFolder(ByVal _
   DirectorySpec As String) As _
   Boolean
   Dim sDirSpec As String
   On Error GoTo MakeFolder_Error
   sDirSpec = Trim$(DirectorySpec)
   If Len(sDirSpec) Then
      MakeFolder = True
      If m_fso.FolderExists( _
         sDirSpec) Then Exit Function
      If Not MakeFolder( _
         StripFileName(sDirSpec)) _
         Then
         MakeFolder = False
         Exit Function
      End If
      m_fso.CreateFolder (sDirSpec)
   End If
MakeFolder_Exit:
   Exit Function
MakeFolder_Error:
   MakeFolder = False
   Resume MakeFolder_Exit
End Function
```

Add the Test Harness

At this point, you need a simple form to act as a test harness for your class, so you can easily test the class's properties and methods (download Listing 2 from DevX). Insert a form in your project and add a TextBox control for the fully qualified path, file name, and buttons to test the class's properties and methods (see Figure 2). Now create an instance of your CFileString class to use on the form:

```
Public MyFile As New CFileString
```

The VB6 FileSystemObject is built into the Microsoft Scripting Runtime library.

First, add code in the Click event of the Load FileString CommandButton to set the PathAndFileName property of the CFileString class from the Text1 TextBox control. Then use the IIf statement to set the check boxes to indicate whether the file and folder exist:

```
Private Sub Command3_Click()
   MyFile.PathAndFileName = Text1
   Check2 = IIf(MyFile.FileExists, _
      vbChecked, vbUnchecked)
   Check3 = _
      IIf(MyFile.FolderExists, _
      vbChecked, vbUnchecked)
   Text2 = ""
   Label2 = "FileString Loaded!!"
End Sub
```

You can add code in the other CommandButton Click events to call each of the CFileString class properties and methods and display the results. For example, the FileType button simply displays the FileType property of the CFileString class:

```
Private Sub Command6_Click()
   Text2 = MyFile.FileType
   Label2 = "File Type:"
End Sub
```

After you complete the code for each of the class's properties and methods, press the F8 key to single-step through the code and test how these new VB6 string functions operate within your class. You might want to extend the class to expose other properties for the File object or to call other methods of the FileSystemObject. When your class is complete, you can add the CLS file to any of your applications or compile it into an ActiveX DLL and reference it in your applications. VBPJ

About the Authors

Chris Barlow is president and CEO of SunOpTech, a developer of manufacturing decision-support applications, including the ObjectBank and the ObjectJob systems. Chris holds degrees from Harvard Business School and Dartmouth College, where he worked with Drs. Kemeny and Kurtz on the Basic language. Reach Chris at Chris@VBExpert.com.

Stan Schultes is lead developer at SunOpTech, where he is responsible for development and worldwide support of the ObjectBank and ObjectOrder products. Stan has 20 years of experience in the computing field, most spent with a Fortune 200 company, where he developed manufacturing systems and was a corporate technology consultant. Stan holds a degree in computer engineering from Purdue University. Reach Stan at Stan@VBExpert.com.

CODE ONLINE

You can find all the code published in this issue of **VBPJ** on The Development Exchange (DevX) at http://www.vbpj.com. For details, please see "Get Extra Code in DevX's Premier Club" in Letters to the Editor.

Manage Strings Easily in VB6
Locator+ Codes
Listings for the entire issue, including Listings 1 and 2 and the simple FileString class and test-harness application (free Registered Level): VBPJ1098
Listings for this article only, plus the code files described above (subscriber Premier Level): GS1098