Put COM Interfaces to Work

by Chris Barlow

Use COM interfaces to build a document management app that converts text files.

Microsoft's Component Object Model (COM) is the foundation for all ActiveX servers. One of COM's most powerful features is its ability to create *interfaces* to a COM object. These interfaces allow you to publish a set of properties and methods that other applications can use to exchange information with your application through COM. Many books and articles have been written about developing COM objects and their interfaces. Unfortunately, most are written for C++ programmers and contain a lot of detail about Globally Unique IDs (GUIDs), the *IUnknown interface*, *CoCreateClass*, and so on.

Visual Basic programmers have it much easier than C++ programmers because VB handles most of these details. Sometimes it is difficult to cut through all this extra information to find out what a VB programmer needs to do to create COM interfaces. It might surprise you that even a beginning programmer can quickly and easily write a COM interface and ActiveX objects to use it.

At my company, we used VB5 to write a document-management product called ObjectBank. You can drag and drop any kind of file or OLE object onto the ObjectTeller and fill out the deposit slip with keywords, which enables you to find the document quickly. In a document management system, it is vital to index the document correctly. But how can a program automatically index so many different file formats? The ObjectBank makes use of the power of COM interfaces to publish an IOBSFileConvert interface. This interface is a set of properties and methods that provides the index information for a document and

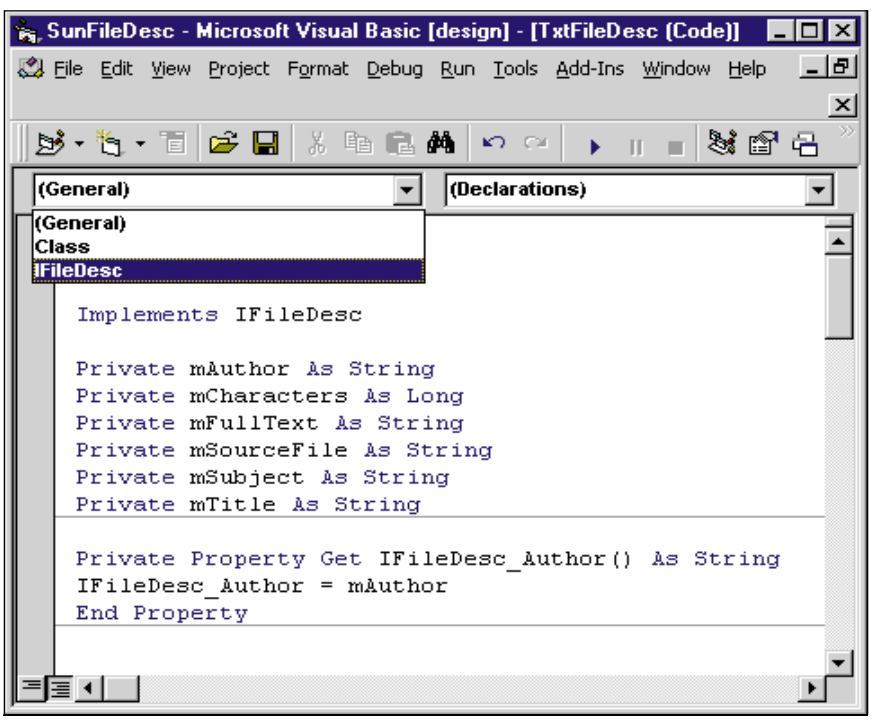

FIGURE 1 ***Add Procedures and Properties with Interfaces.*** *You get a special set of procedures in your code window when you implement a referenced interface. These procedures let you implement all the properties and methods of the interface.*

Chris Barlow is president and CEO of SunOpTech, a developer of manufacturing decision-support and supply chain applications, including the ObjectBank, ObjectOrder, and ObjectJob Systems. He holds two U. S. Patents related to software for decentralized distributed asynchronous object-oriented and scheduling systems. Chris, who is a frequent speaker at VBITS, Tech•Ed, and DevDays and has been featured in two Microsoft videos, holds degrees from Harvard Business School and Dartmouth College. Reach Chris at ChrisB@SunOpTech.com or through SunOpTech's Web server at www.SunOpTech.com.

automatically adds the keywords to the deposit slip.

This article shows you how to prepare a slightly simplified version of this interface, called IFileDesc. You can use this as a template to create your own COM interfaces. You need to create three VB projects: one for the IFileDesc interface, one to implement the IFileDesc interface for a specific type of file, and one "test harness" to emulate the functions of the ObjectBank.

First, create the IFileDesc interface. You can contain this interface within a type library, but it is easier for you as a VB programmer to contain this interface in a simple ActiveX DLL. The DLL will have two classes, but no code. You'll use the DLL to publish the standard properties and methods of the IFileDesc class.

Next, start VB5 and create a new ActiveX DLL project. Give the new project a public class called Class1. Every ActiveX DLL needs at least one public class, so rename this class to EmptyPublicClass. Insert another class module, name it IFileDesc, and set its instancing property to PublicNotCreatable. The user will never create an instance of this class—it is only used as the interface that other classes implement. Insert this code in this class module for the public properties and methods of this interface:

```
Option Explicit
Public SourceFile As String
Public Title As String
Public Subject As String
Public Author As String
Public Characters As Long
Public FullText As String

Public Function GetContent() As Boolean

End Function
```

Name the project FileDesc, save it, and compile it as

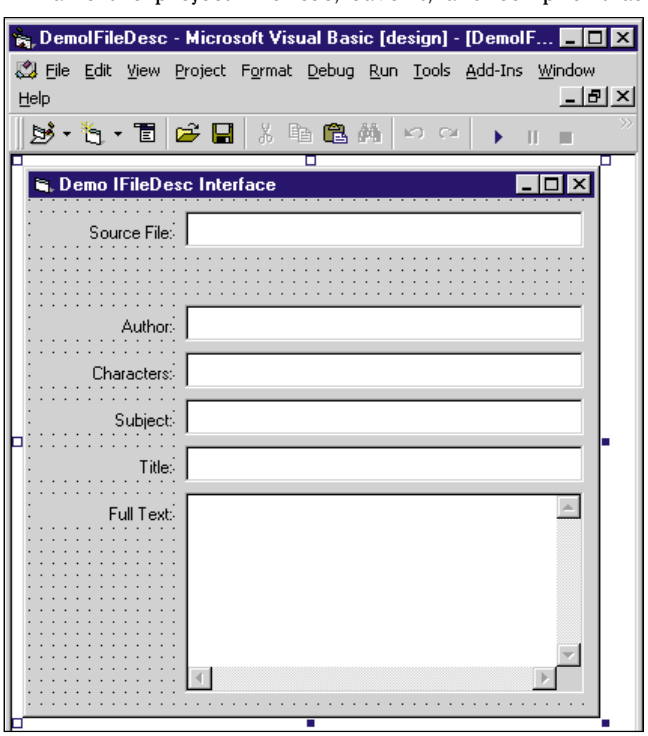

FIGURE 2 ***Create a Test Harness Form.*** *Every ActiveX object that you develop needs to have a test harness so you can test the properties and methods of your component. This form implements the OLEDragDrop event so you can drag files from Windows Explorer to the form.*

FILEDESC.DLL. You've completed the COM interface.

Now create a file converter that implements this COM interface to index memos written in a standard text file format:

Date:	*November 17, 1997*
Author:	*Chris Barlow*
Subject:	*IFileDesc Interface*
Title:	*Test Text File for Column*

This is a test text file for the FileDesc interface column.

The file converter opens this text file, searches for the Date, Author, Subject, and Title, and returns them through the IFileDesc interface.

Create a second VB ActiveX DLL project, and name the class module TxtFileDesc. Select References from the Project menu, and add a reference to your FileDesc DLL. This reference lets you implement the IFileDesc interface:

```
Option Explicit
Implements IFileDesc

Private mAuthor As String
Private mCharacters As Long
Private mFullText As String
Private mSourceFile As String
Private mSubject As String
Private mTitle As String
```

Click on the left combo box in the code window, and you'll see an entry for the IFileDesc interface (see Figure 1). Next, add code for the properties and methods of this interface. COM requires every object that implements an interface to implement every property and method of that interface. You need to click on each of the items in the right-hand combo box to add all the properties and methods for the IFileDesc interface to the code window. Note that the names of these procedures are prefaced with the name of the interface being implemented, allowing your project to implement different interfaces that have the same name for a property or method.

Most of the properties of this interface are read-only∇that is, you only need to add code in the Property Get procedure, and you can leave the Property Let procedure empty. Add code for all the properties using the Author property as a guide (for the complete code, see Listing 1 on the free, Registered Level of The Development Exchange):

```
Private Property Get IFileDesc_Author() As String
IFileDesc_Author = mAuthor
End Property

Private Property Let IFileDesc_Author(ByVal RHS As String)
End Property
```

The SourceFile property is a special case because it needs code for both its Get and Let procedures. The calling application uses this property to pass the name of the file to be converted. This enables you to write and read the file:

```
Private Property Let IFileDesc_SourceFile(ByVal RHS As _
   String)
mSourceFile = RHS
End Property

Private Property Get IFileDesc_SourceFile() As String
```

```
IFileDesc_SourceFile = mSourceFile
End Property
```

The GetContentMethod performs the real work in the converter. This procedure opens the text file and parses the content to locate the Subject, Author, and Title. It also stores the full text of the file in the FullText property, and the number of characters in the Characters property. This procedure also checks the length of the private mSourceFile variable to make sure this property has been set, then opens the file for binary read and gets a free file handle with the FreeFile statement. Note that the Input statement can read the entire file contents into a buffer variable by using the LOF function to get the length of the file. Next, the method sets the FullText property to this input buffer and uses the Instr and Mid functions to search for keywords in the text. Finally, the method sets the Characters property to the length of the text and sets the method to return True:

```
Private Function _
   IFileDesc_GetContent() As Boolean
Dim fn As Integer, buf As String
If Len(mSourceFile) Then
   fn = FreeFile
   Open mSourceFile For Binary As #fn
   buf = Input(LOF(fn), fn)
   mFullText = buf
   mSubject = Mid(mFullText, InStr_
      (mFullText, "Subject:") + 9, 20)
   mAuthor = Mid(mFullText, InStr_
      (mFullText, "Author:") + 8, 15)
   mTitle = Mid(mFullText, InStr_
      (mFullText, "Title:") + 7, 40)
   mCharacters = Len(mFullText)
   IFileDesc_GetContent = True
End If
End Function
```

Name the project TextFileDesc, save it, and compile it as TXTFILEDESC.DLL. You've now written a COM object file converter that implements your IFileDesc COM interface. Next, you need to write a test harness—a small application that provides just enough functionality to test your ActiveX objects. Start a new VB Standard EXE project and name it DemoIFileDesc. Select References from the Project menu and add a reference to your FileDesc DLL, so you can call ActiveX objects that implement this interface. Add label and text-box controls to the form for the properties of the IFileDesc interface (see Figure 2).

Press F7 to display the code window for the form, and add code. The first line of code defines a private collection variable to hold the class name for your file converters and the file extensions of the types of file they handle. The file extension serves as the key to the collection. Your test harness can have only one file converter for each file extension, and your procedure must be able to locate the proper file converter's class name by using the file extension as a key. Use the Add method of the collection to add the text file converter to your test harness. You would store this information in the Registry in a real application. When your user registers a new file converter, your app loads it from the Registry:

```
Private mFileDescs As New Collection

Private Sub Form_Load()
mFileDescs.Add _
   "SunFileDesc.TxtFileDesc", "TXT"
End Sub
```

The functionality for your test harness goes in the form's OLEDragDrop event. This event fires when you drag one or more files from the Windows Explorer, Desktop, or any OLE-compatible application. The event passes a DataObject that contains a Files collection, which stores the file names and paths of the dropped files.

The trick to using a COM interface from VB is to define two variables—one for the interface (iConv), and one for the COM object that implements the interface (oConv). You can early-bind the iConv variable to the IFileDesc interface because you reference it in your project. You must dimension the oConv variable as "Object," however, so you can late-bind it to many different file converters. Your application does not know which file converters are available until it loads them from the Registry:

```
Private Sub Form_OLEDragDrop(Data As _
   DataObject, Effect As Long, Button _
   As Integer, Shift As Integer, X As _
   Single, Y As Single)
Dim SourceFile, Ext$, i%
Dim oConv As Object
Dim iConv As IFileDesc
```

This next section of code sets up a For…Next loop to loop through each file in the DataObject's Files collection and determine its file extension:

```
On Error Resume Next
For Each SourceFile In Data.Files
   Text1(0) = SourceFile
   For i = Len(SourceFile) To 1 Step -1
      If Mid(SourceFile, i, 1) = "." _
         Then
         Exit For
      Else
         Ext = Mid(SourceFile, i, 1) _
            & Ext
      End If
   Next
```

Next, the procedure creates an instance of the proper file converter object by using the file extension as a key to the mFileDesc collection. It uses the CreateObject statement to get the text class name of the appropriate file converter so it can be passed as an argument:

```
Set oConv = _
   CreateObject(mFileDescs(Ext))
If Err Then Exit Sub
```

Now set the iConv variable equal to the oConv variable. This allows you to call the methods and properties of the loaded file converter pointed to by oConv from the iConv IFileDesc interface. Set the file name to the SourceFile property and call the GetContent method of the IFileDesc interface. Retrieve the properties from the interface, and set the text controls on your test harness form:

```
   Set iConv = oConv
   iConv.SourceFile = SourceFile
   iConv.GetContent
   Text1(1) = iConv.Author
   Text1(2) = iConv.Characters
   Text1(3) = iConv.Subject
   Text1(4) = iConv.Title
   Text1(5) = iConv.FullText
   Set iConv = Nothing
   Set oConv = Nothing
Next
End Sub
```

That's all there is to it. Now you can extend your test harness to handle other types of files by simply adding another class to TXTFILEDESC.DLL or by creating a new ActiveX DLL. Try out your new procedure on some of your files and see how easy it is to create COM interfaces to your applications.

Code Online

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

Put COM Interfaces to Work

Locator+ Codes

Listings for the entire issue, plus code for the VB procedures for the IFileDesc interface (free Registered Level): VBPJ0298
Listings for this article only, plus the code files mentioned above (subscriber Premier Level): GS0298