Create Programmable Applications

by Chris Barlow

Make your apps more flexible and powerful.

Despite your best efforts, an application will never meet all the needs of all your users. Most often, users request a change to how information is reported. Report generators, such as Seagate Crystal Reports, help ease the load by allowing you to easily change an existing report or create a new report. Three-tier application design also helps you address your users' needs more easily. As long as the interfaces don't change, you can modify a business object that encapsulates a set of business rules to use different rules without affecting the other objects included in the application. Once you implement a solid three-tier design, less experienced programmers can make changes to individual business objects.

But suppose users need to change how data is input into your application or how it interfaces to another application. How do you extend the life of your application so that even these changes don't make it obsolete? Make it programmable!

When you make an application programmable, you recognize that you can't predict all its possible uses. Exposing the functionality of the application to other programs allows another programmer to use it in ways you can't predict.

For example, when Visual Basic was first released, I wrote an application, called CatNav, to help users quickly find the price of a part during a customer phone call. CatNav displayed a form where the user could type in a part number, click on the Find button to look up all the components of that part in a database, and calculate the price. Then the user could click on the Save button to save the part number and price to an ASCII file. Although it was not in the specification, I implemented a DDE interface (ActiveX was not available in 1992!) with two different DDE messages: "Find [partnumber]," which loaded the part number into the text box and clicked on the Find button, and "Save," which clicked on the Save button.

CatNav worked well when users needed to find the price for only a few parts. One month later, though, a user told me he wanted to look up prices and type them into a letter using Word. I was able to quickly write a Word macro that used DDE to pass a part number from Word to CatNav and return the price. A few weeks later, another user told me he had a database of part numbers and wanted to calculate the price for each of them. I wrote a tiny Visual Basic application that read the database, used DDE to link to CatNav to calculate the price, then wrote the calculated price back to the external database.

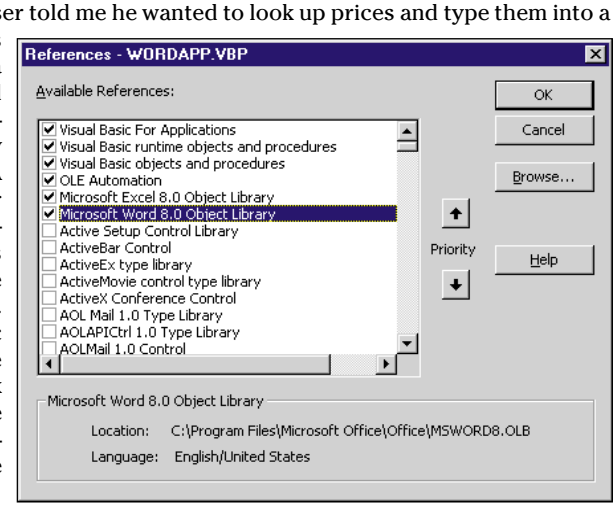

FIGURE 1 ***Reference Word and Excel.*** *When you set your project's references, you instruct Visual Basic to load the type libraries for these applications and early-bind your application's objects to the specific classes. Use the built-in Object Browser to scan all the objects exposed by these applications.*

CatNav was more than a standalone application. Through its DDE programmability, CatNav became a business object that knew how to do price cal-

Chris Barlow is president of SunOpTech, a developer of document-management, 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 Chris@VBExpert.com or through his Web server at http://www.VBExpert.com.

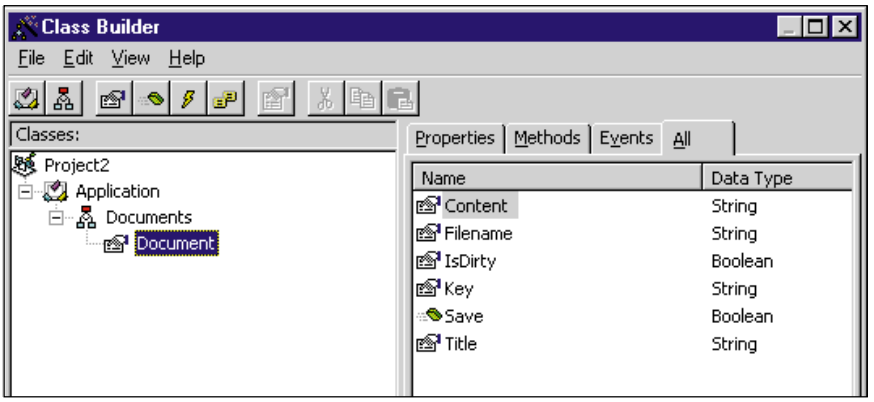

FIGURE 2 ***Class Builder Utility Add-In.*** *The Class Builder makes it easy to add new classes to your application and set the properties, methods, and events. Take a look at the code generated by this add-in for a good start on writing your own code for your classes.*

culations and return them to any application. ActiveX makes this kind of development much simpler.

EXPERIMENT WITH WORD AND EXCEL

If you have Microsoft Office 97 on your computer, you can experiment with some of the neatest programmable applications—Word and Excel. Both of these applications export a complete object model with objects your applications can use as components. You can view these object models, along with many others, on the VBA Objects Web site at http://objects.windx.com.

Start a new Visual Basic Standard EXE project and click on References under the Project menu to add the Microsoft Word 8 Object Library and the Microsoft Excel 8 Object Library (see Figure 1). Put a CommandButton control on the form, and add this code in the Click event to create a new Application object within Word:

```
Private Sub Command1_Click()
Dim App As New Word.Application
With App.Documents.Add
   .Content.Text = _
      "This is a test of Word"
   MsgBox .Words(1)
End With
App.Visible = True
End Sub
```

As with most programmable applications, the top-level Application object is the starting point for any functionality. Use the Application object to create any other objects in the object model. In this case, use the Add method of the Application object's Documents collection to add a new document. Then add the text "This is a test" to the document using the Text property of the Content range of the new document. You can create a message box containing the first word in the document by displaying the first item in the document's Words collection. Finally, set the Application object's Visible property to True.

Press F8 to single step through this code. Notice that Word is running when you set the text into the document, but you don't see it on your screen because you started Word as an ActiveX server. As such, it doesn't display any user interface until you set the Visible property to True. After you execute the last line of code, you'll see the new document with the text in the right place.

HOW DO YOU EXTEND THE LIFE OF YOUR APPLICATION SO THAT CHANGES DON'T MAKE IT OBSOLETE? MAKE IT PROGRAMMABLE!

Add another CommandButton control to your form and try this code with Excel. Notice that some of the objects are different, but you still start with an Application object:

```
Private Sub Command2_Click()
Dim App As New Excel.Application
With App.Workbooks.Add
   With .Worksheets(1)
      .Range("A1").Formula = _
         "This is a test of Excel"
      MsgBox .Range("A1").Text
   End With
End With
App.Visible = True
End Sub
```

When creating your own programmable applications, you'll find some things about Visual Basic actually make your job harder. For example, when designing an ActiveX DLL, you begin with the object model and plan the properties and methods of your classes. But when you design an application with a user interface, Visual Basic makes it easy to drag a control to a form, double-click, and start writing code. If you want to turn off the display of the form's status bar, simply click on the View Status Bar menu item and type this code:

```
Sub mnuViewPnl_click
   pnlStatus.Visible = Not _
      pnlStatus.Visible
End Sub
```

You've just created a problem if you're trying to develop a programmable application. How can an external program change the display of your application's status bar? If you really want to create a programmable application, your Application object should have a Boolean ShowStatusBar property. This property should have Property Let and Get procedures contained within the Application class, like the procedures below, which toggle the Visible property of the status bar:

```
Public Property Let ShowStatusBar( _
   ByVal vData As Boolean)
   mvarForm.pnlStatus.Visible = vData
End Property

Public Property Get ShowStatusBar () _
   As Boolean
   ShowStatusBar = _
      mvarForm.pnlStatus.Visible
End Property
```

In your menu item code on the form, use this same ShowStatusBar property of the Application object. This way, the user interface built into your application interacts with your application through the Application object:

```
Sub mnuViewPnl_click
   MyApp.ShowStatusBar = Not _
      MyApp.ShowStatusBar
End Sub
```

Creating a programmable application requires the same up-front effort as designing an ActiveX DLL. You can't just drag controls to a form and double-click to write code. You need to design the

object model for your application, plan the properties and methods for each of your object classes, and make sure all the functionality of the user interface correlates to properties and methods of your application's classes.

MAKE YOUR APP PROGRAMMABLE

How do you convert a Standard EXE design into an ActiveX programmable EXE? Let's take it step by step, beginning with a simple multiple document interface (MDI) text editor. I'll touch on some of the steps necessary to convert this simple text editor to a programmable application. You can download the complete project with all these examples from the free, Registered Level of The Development Exchange (see the Code Online box for details).

The Visual Basic Application Wizard quickly builds the code for the simple text editor. Launch the Application Wizard and step through the screens to build an MDI application (or download the source from The Development Exchange). Let's take a look at the StdTextEdit project to understand how it works. First, there are no classes in the standard project because it is not an ActiveX project. Its Sub Main procedure simply shows the main form:

```
Sub Main()
   Set fMainForm = New frmMain
   fMainForm.Show
End Sub
```

The main form's Load event then loads a new document form using the LoadNewDoc procedure while maintaining a static count of the number of documents created:

```
Private Sub LoadNewDoc()
   Static lDocumentCount As Long
   Dim frmD As frmDocument
   lDocumentCount = lDocumentCount + 1
   Set frmD = New frmDocument
   frmD.Caption = "Document " & _
      lDocumentCount
   frmD.Show
End Sub
```

Now think about converting this application into a programmable one. Conversion can be difficult because you need to change some of the basic functionality of the application; you're usually better off starting from scratch. Here, however, I'll have room only to show how to convert the standard application rather than start over with a new application. You'll quickly see the differences and gain a better understanding of the entire process.

First, plan the object model. You need to add an Application class as the top-level object in your application. The Application class will contain several properties and methods to let a program interact with your application. For example, you can create a Visible property that works like Word or Excel's property, or you can create the ShowStatusBar property discussed previously. The Application class should also contain a Documents class as a collection of open documents and an ActiveDocument property that returns the current document (see Figure 2).

Copy your standard EXE code to a new folder or download the ActiveTextEdit project from The Development Exchange. Make sure your project is set to an ActiveX EXE rather than a Standard EXE. Then add the Application and Documents classes. Use the Class Builder Utility add-in to create these classes; it adds some of the property creation code for you (download Listing 1 from the free, Registered Level of The Development Exchange).

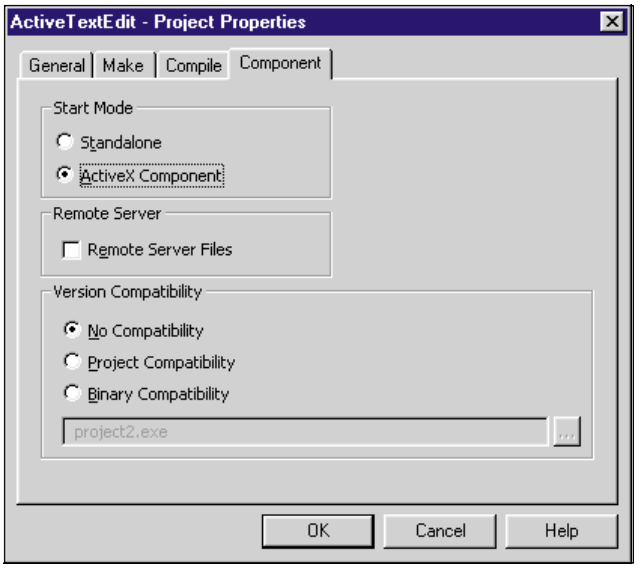

FIGURE 3 ***ActiveX Component.*** *When you change your project property to ActiveX Component, you can see what happens when another application instantiates one of your classes. Visual Basic recognizes that the Class Builder makes it easy to add new classes to your application and set the properties, methods, and events.*

You need to make a change in the Main procedure. The App object of an ActiveX EXE includes a StartMode property that indicates whether the application was started as a standalone EXE or as a component. If your application is started by another program as an ActiveX component, it uses code like this:

```
set MyApp = New _
   ActiveTextEdit.Application
```

Your application's Main procedure executes when the other program uses a property or method of the Application class. However, if the user simply double-clicks on your application to run it standalone, your Main procedure needs to create the Application object. Rather than simply showing the main form in your Main procedure, check the App.StartMode property and, if standalone, create a new instance of the Application class and set its Visible property to True:

```
Sub Main()
   Select Case App.StartMode
   Case vbSModeStandalone
      Set MyApp = New Application
      MyApp.Visible = True
   End Select
End Sub
```

The Application Class_Initialize procedure creates the Documents class and loads the main form. After the main form is loaded, it calls the LoadNewDoc procedure, which you change to call the Add method of the Documents class.

If you were writing this application from scratch, you'd probably have a separate Document class so your application wouldn't need to load a form when it

operates as a component. However, because we're using this project to demonstrate the conversion to a programmable application, you can use the frmDocument form as your Document class. Remember, a form is a lot like a class. In VB5, you can even add your own properties and methods to a form. Add these Public properties to the document form:

```
Public Key As String
Public Title As String
Public IsDirty As Boolean
Public Filename As String
```

Then add these properties procedures to set or return the Content property of the document from the Text property of the RichTextbox control:

```
Public Property Get Content() As String
   Content = rtfText.Text
End Property

Public Property Let Content(vData$)
   rtfText.Text = vData
End Property
```

Although the conversion process has just begun, you have enough working code to test your application. Always practice incremental development—get enough code done so the application works, debug it, then code some more. Run your ActiveTextEdit project in single-step mode and watch it create the Application and Documents classes. Does it work in standalone mode?

Now change the project properties to run it as an ActiveX component (see Figure 3), and start another instance of Visual Basic to create a test project. Set a reference to the ActiveTextEdit application in the project references. Put a CommandButton control on the form and add this code:

```
Private Sub Command1_Click()
Dim MyApp As New Application
MsgBox MyApp.Documents.Count
MyApp.ActiveDocument.Content = _
   "This is a test"
MyApp.ShowStatusBar = False
MyApp.Visible = True
End Sub
```

Single step through the code and you'll see that your Application object is created, the Documents class is initialized with a single document, and your application is not visible on the screen. After you set the Visible property to True, you'll see your text in the first document. You've now created a programmable application. ×

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.

Create Programmable Applications
Locator+ Codes

Listings for the entire issue, plus the simple text editor application built using the Application Wizard and the partially converted active text editor (free Registered Level): VBPJ0998

Listings for this article only, plus the code described above and the more comprehensive conversion to an active text editor (subscriber Premier Level): GS0998