Track Code Changes

by Chris Barlow

Create an add-in to flag changes with programmers' initials and change date.

Click & Retrieve Source CODE!

As you do more Visual Basic programming, you find yourself making changes to procedures or applications that you or someone else in your organization wrote a while ago. Maintaining a growing code base of existing applications can be one of the most time-consuming and error-prone activities of a software development organization. To make this process flow more smoothly at my company, we instituted coding standards so each code change is flagged with the programmer's initials and change date. This standard has made it easier to track changes to the code base or discuss a prior change with the person who made it.

When you make multiple changes to a program in a single session, it takes discipline to type *"CRB 05/15/98"* with each change. It's easy to skip this step and go on to the next code change with the rationalization that it's only a minor change or that you'll come back and enter your initials later, after you've debugged the code. Predictably, you never get back to flag the change. Too bad Visual Basic doesn't have an option to insert your initials at the press of a toolbar button. What was Microsoft thinking?

But wait! Microsoft *was* thinking after all—the entire Visual Basic integrated development environment (IDE) is extensible through add-ins. Any COM object that implements the IDTExtensibility COM interface can hook into the IDE and seamlessly extend the development environment.

Add-ins were available in Visual Basic 4.0 in a limited manner, but with version 5, you can create more meaningful add-ins. For more details, take a look at Francesco Balena's article, "Roll Your Own Add-Ins" [*VBPJ* August 1997] and Sam Patterson's COMponent Builder columns, "Add-Ins Save You Time, Effort" [*VBPJ* July 1997] and "Add Some Zip to Your Add-Ins" [*VBPJ* October 1997]. Despite these resources, my e-mail indicates that many people think writing an add-in is over the head of someone getting started with Visual Basic. Not true! You can write an add-in that marks your changes by inserting your initials, along with the date and time, directly into your Visual Basic code window with fewer than 20 lines of code. In the process, you'll learn about the IDE's extensibility object model and how to add an icon for your add-in to one of Visual Basic's toolbars.

Let's get started. You need VB5 to work with the code for this column. You can download the complete code for this app from the free, Registered Level of The Development Exchange (see the Code Online box for details), but with a project this small, it's a great learning experience to start from scratch (see Listing 1). Start a new ActiveX Visual Basic project and change the name of the class module to Connect. Select Project References and add a reference to Microsoft Visual Basic 5.0 Extensibility (which contains the IDTExtensibility COM interface) and to the

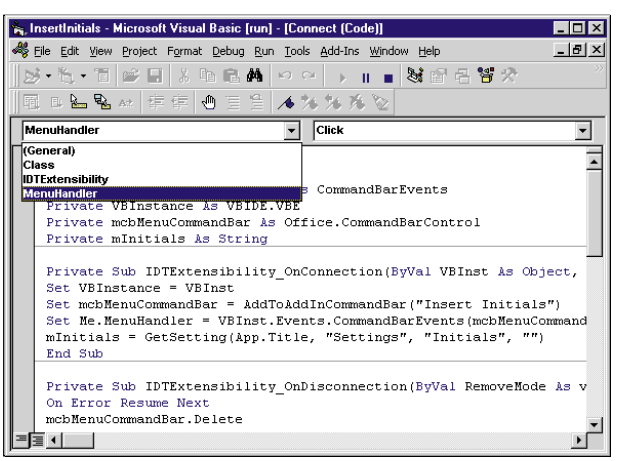

FIGURE 1 ***Connect Class.*** *When you implement a COM interface or declare an ActiveX component WithEvents, your code window shows additional objects where you can add code.*

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.

Microsoft Office 8.0 Library (which contains the Visual Basic menus and toolbars as Office CommandBar objects). Change the Project name to InsertInitials. Type in these five lines of code:

```
Implements IDTExtensibility
Public WithEvents MenuHandler As _
   CommandBarEvents
Private VBInstance As VBIDE.VBE
Private mcbMenuCommandBar As _
   Office.CommandBarControl
Private mInitials As String
```

The first line shows that this COM object (remember, you are creating an ActiveX EXE, which is a COM object) will implement the Visual Basic IDE extensibility interface. The second line sets MenuHandler as the procedure that will respond to the events of the CommandBar object, such as a mouse click on a toolbar icon. The WithEvents statement, new to VB5, lets you respond to events in another ActiveX object. The MenuHandler procedure needs to be Public because the VB IDE calls it when the user clicks on your toolbar button. The third line saves the instance of VB that is passed to your add-in. The fourth line saves a reference to the toolbar where you add an icon for your add-in, so you can remove the icon if the user unloads the add-in. Finally, the last line stores the user's initials.

Look at the code window for your Connect class (see Figure 1). In addition to the normal General and Class items in the Object combo box, the IDTExtensibility and MenuHandler objects are listed. Here you add code to implement the IDE's COM interface and handle a click on your toolbar icon. Remember, a COM object that implements a COM interface *must* implement all the properties and methods of that interface. Click on the IDTExtensibility object, then click on each of the four items in the Procedure combo box. These four items are the only methods of this COM interface. Although all procedures must be present in your class, you need to add code only to the OnConnection and OnDisconnection methods.

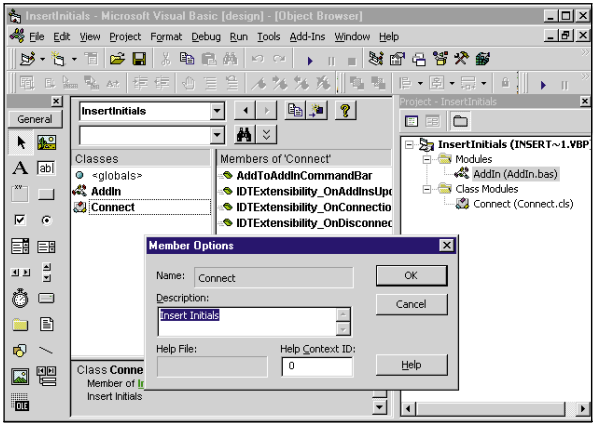

FIGURE 2 ***Class Description.*** *The Add-In Manager displays the class Description property—but it's hard to find out where to change that description. You can only change it from the Object Browser by right-clicking to display the class properties.*

The IDTExtensibility_OnConnection method is called when Visual Basic instantiates your add-in—either because the user selected it from the list of available add-ins, or because the user selected it the last time Visual Basic was run, causing it to load again

VB5

```
Option Explicit
Implements IDTExtensibility
Public WithEvents MenuHandler As CommandBarEvents
Private VBInstance As VBIDE.VBE
Private mcbMenuCommandBar As Office.CommandBarControl
Private mInitials As String
Private Sub IDTExtensibility_OnConnection _
  (ByVal VBInst As Object, ByVal ConnectMode As _
  vbext_ConnectMode, ByVal AddInInst As VBIDE.AddIn, _
  custom() As Variant)
Set VBInstance = VBInst
Set mcbMenuCommandBar = _
  VBInstance.CommandBars("Edit").Controls.Add
mcbMenuCommandBar.Caption = "Insert Initials"
Clipboard.SetData LoadPicture(App.Path & "\icon.bmp")
mcbMenuCommandBar.PasteFace
Set Me.MenuHandler = _
  VBInstance.Events.CommandBarEvents _
  (mcbMenuCommandBar)_mInitials = _
  GetSetting(App.Title, _
  "Settings", "Initials", "")
End Sub
Private Sub IDTExtensibility_OnDisconnection _
  (ByVal RemoveMode As vbext_DisconnectMode, custom() _
  As Variant)
mcbMenuCommandBar.Delete
End Sub
Private Sub _
 IDTExtensibility_OnStartupComplete(custom() _
 As Variant)
'
End Sub
Private Sub IDTExtensibility_OnAddInsUpdate(custom() _
 As Variant)
'
End Sub
Private Sub MenuHandler_Click(ByVal CommandBarControl _
  As Object, handled As Boolean, CancelDefault As _
  Boolean)
Dim SLine&, SCol&, ELine&, ECol&
With VBInstance.ActiveCodePane
  .GetSelection SLine, SCol, ELine, ECol
  .CodeModule.InsertLines SLine, "'" & _
     mInitials & " " & Format$(Now, _
     "dd-mmm-yy hh:mm:ss")
End With
End Sub
```

LISTING 1 ***Add-In Connect Class.*** *This class only has 11 lines of code, yet it adds useful functionality to the Visual Basic development environment. Now whenever you make a change to your source code, you can add your initials and the date and time so these changes will be easy to track.*

at startup. The majority of your code resides in the IDTExtensibility_OnConnection method; this will be almost the same for every add-in you write. In this procedure, save the instance of Visual Basic that is passed as an argument, add your add-in to one of the Visual Basic menus or toolbars, and load the user's initials from the Registry. I chose to add my add-in to Visual Basic's Edit toolbar, so I used the Add method of the Controls collection of that toolbar, saved the reference, then set the caption.

ADD AN ICON TO YOUR TOOLBAR

You won't need an icon if you're adding your add-in to a menu, but you'll need an icon when adding your add-in to a toolbar. Because your project does not have a form or resource file, you add an icon to the toolbar using the LoadPicture statement to load a 16-by-16 bitmap to the Clipboard, then using the PasteFace method of the CommandBarControl object. I used MSPaint to create a simple icon bitmap, which is included with the source code you can download from the free, Registered Level of The Development Exchange (see the Code Online box for details). Link your MenuHandler procedure to the events for this button. Finally, load the user's initials from the Registry—later you'll see how to get them into the Registry when your add-in is registered with the Visual Basic Add-In Manager:

```
Private Sub _
   IDTExtensibility_OnConnection _
   (ByVal VBInst As Object, _
   ByVal ConnectMode As _
   vbext_ConnectMode, _
   ByVal AddInInst As VBIDE.AddIn, _
   custom() As Variant)
Set VBInstance = VBInst
Set mcbMenuCommandBar = _
   VBInstance.CommandBars("Edit"). _
   Controls.Add
mcbMenuCommandBar.Caption = _
   "Insert Initials"
Clipboard.SetData LoadPicture _
   (App.Path & "\icon.bmp")
mcbMenuCommandBar.PasteFace
Set Me.MenuHandler = _
   VBInstance.Events.CommandBarEvents _
   (mcbMenuCommandBar)
mInitials = GetSetting(App.Title, _
   "Settings", "Initials", "")
End Sub
```

One line of code in the IDTExtensibility_OnDisconnection method removes the CommandBar button when the user no longer wants to use the add-in:

```
Private Sub _
   IDTExtensibility_OnDisconnection _
   (ByVal RemoveMode As _
   vbext_DisconnectMode, custom() _
   As Variant)
mcbMenuCommandBar.Delete
End Sub
```

Now that you've connected your add-in to VB, added it to the toolbar, and linked it to the toolbar's Button event, you need to add the code in the MenuHandler procedure to insert the user's initials. If you scan the VB IDE extensibility object model in VB's Object Browser, you'll see an ActiveCodePane property that returns a CodePane object—the code window where the user enters code. Use the GetSelection property of the CodePane object to find which lines of code are selected. You want to insert the user's initials just above the start line of the selection. You can't insert a line of code in a code module using the CodePane object; you must use the InsertLines method of the Code Module object:

```
Private Sub MenuHandler_Click _
   (ByVal CommandBarControl As _
   Object, handled As Boolean, _
   CancelDefault As Boolean)
Dim SLine&, SCol&, ELine&, ECol&
With VBInstance.ActiveCodePane
   .GetSelection SLine, SCol, ELine, _
      ECol
   .CodeModule.InsertLines SLine, _
      "'" & mInitials & " " & _
      Format$(Now, _
      "dd-mmm-yy hh:mm:ss")
End With
End Sub
```

TELL VISUAL BASIC ABOUT YOUR ADD-IN

That's all the code you need for your add-in class. Before you can test the code, you need to let the Visual Basic Add-In Manager know about your new add-in. The Add-In Manager uses the *Add-Ins32* section of the *vbaddin.ini* file to get a list of add-ins. You could edit this file manually to add a reference to your add-in, but because you're developing an ActiveX EXE, there's a better way. Add some code to the Add-In Manager's Sub Main startup procedure so that run-

ning your add-in completes the OLE registration and sets up the INI files and Registry. Your user only needs to run this EXE once to register it and set his or her initials.

Add a module and name it AddIn.bas. You need the declaration for the WritePrivateProfileString API call so you can update the vbaddin.ini file. Changing your Project properties so the Startup Object is Sub Main executes this procedure whenever the project starts. Use the App object's StartMode property to make sure your add-in runs as a standalone EXE and is not started by Visual Basic when establishing an add-in connection:

```
Declare Function _
   WritePrivateProfileString& Lib _
   "Kernel32" Alias _
   "WritePrivateProfileStringA" _
   (ByVal AppName$, ByVal KeyName$, _
   ByVal keydefault$, ByVal FileName$)
Sub Main()
Dim txt$
If App.StartMode = _
   vbext_psm_StandAlone Then
   txt = GetSetting(App.Title, _
      "Settings", "Initials", "")
   txt = InputBox("Enter Initials", _
      "InsertInitials VB Add-In", txt)
   SaveSetting App.Title, "Settings", _
      "Initials", txt
   WritePrivateProfileString _
      "Add-Ins32", _
      "InsertInitials.Connect", "0", _
      "vbaddin.ini"
   MsgBox "Insert Initials Add-In " & _
      "Registered"
End If
End Sub
```

There's one last trick you need to use with your Connect class. The Add-In Manager uses the Description property of the class to display the add-in's name in its window. You need to set the Description property to make your class appear in the proper place in the list. You can't set a class description from the Properties window—you need to view the class in the Object Browser, right-click to display the class properties, and enter the description (see Figure 2).

Now compile your add-in, run the EXE, and enter your initials. Go back to the Visual Basic development environment, click on the Add-Ins menu to display the Add-In Manager, and enable your add-in by checking it. When you click on the OK button, the Add-In Manager instantiates your add-in and calls the Connect method. You should see your new button appear on the Edit toolbar. Use the Toolbars menu item from the View menu if the Edit toolbar does not display. Open a code window and click on your button—you should see your initials and the current date and time. Cool! You've extended Visual Basic to add the desired functionality. Imagine what else you could do.

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.

Track Code Changes

Locator+ Codes

Listings for the entire issue, plus the source code, toolbar bitmap image, and compiled DLL needed to start using the add-on with VB (free Registered Level): VBPJ0698

Listings for this article only, plus the code described above (subscriber Premier Level): GS0698