Visual Basic Concepts

Visual Studio 6.0 1 out of 1 rated this helpful

The Visual Basic IDE as an ActiveX Document Container

Using the Extensibility object model and add-ins, you can greatly extend the Visual Basic development environment. You can also create ActiveX documents to extend the Visual Basic IDE. Develop your ActiveX document as you would any other, but use the CreateToolWindow function to create a tool window in the IDE which will contain the ActiveX document.

Note This topic assumes you have some knowledge of add-ins — what they are, and how to create them. If you are unfamiliar with add-ins, see "Add-Ins Overview," and "Creating a Basic Add-In," in "Extending the Visual Basic Environment with Add-Ins."

Using the CreateToolWindow function has the following advantages over other add-ins:

- The CreateToolWindow function returns a dockable window.
- The window correctly follows window minimize and maximize semantics without any coding. For example, tool windows will "remember" where they were last placed or docked.

The CreateToolWindow Function

The CreateToolWindow function is a method of the Windows collection. The function creates a tool window — a container for the ActiveX document — and returns a reference to the window. The following series of steps outline what happens in a typical scenario:

1. On the Add-Ins menu, the user clicks AddIn Manager, and a list of all available add-ins appears.
2. From the list, the user clicks "MyActiveXAddIn."
3. As the add-in is created, the OnConnect event occurs.
4. In the OnConnect event, the code invokes the CreateToolWindow function, creating the tool window.
5. The function call returns a reference to the newly created tool window that contains the ActiveX document.
6. Using the reference, the code can manipulate tool window properties. For example, to show the tool window, the Visible property is set to True.

To create a simple Add-in using an ActiveX document

1. On the **File** menu, click **New Project** to open the **New Project** dialog box. Double-click the **AddIn** icon to create a new Add-in project.
2. Click the **Project** menu, then click **Add UserDocument** to open the **Add UserDocument** dialog box. Double-click the **UserDocument** icon to add a UserDocument to the project.
3. In the **Properties** window, double-click **Name**, and change the name of the UserDocument to "axdUserDoc."
4. In the **Project Explorer** window, double-click the **Connect** icon to open the class module's code window. There is already a fair amount of code written in the template, and we will just add a bit to

it. Look at the Declarations section, and modify it to resemble the following code:

```
Implements IDTExtensibility

Public FormDisplayed         As Boolean
Public VBInstance             As VBIDE.VBE
Public mcbMenuCommandBar As Office.CommandBarControl
Dim frmAddIn                  As New frmAddIn
Public WithEvents MenuHandler As CommandBarEvents

' Add the following declarations for the
' CreateToolWindow function:
Private mWin As Window
Private mobjDoc As axdUserDoc
Const guidMyTool$ = "(4244B234-E45F-12dg-8O3f-04884)"
```

The code you added declares the necessary object variables for the CreateToolWindow function. The Window variable "mWin" will be set to the reference returned by the function; you can then use this reference to show or hide the window. The second variable, "mobjDoc," will be set to reference the UserDocument. You can then use this reference to manipulate the ActiveX document. Finally, the constant "guidMyTool" is used by the function to identify the window instance; for every tool window you create, this string must be unique.

5. In the code window, scroll down to the OnConnection event. We will add code to this event, but to avoid confusion, select all of the code in the event, and delete it. Then add the following code:

```
Private Sub IDTExtensibility_OnConnection(ByVal VBInst As Object, ByVal ConnectMode As
    Set mWin = AddInInst.VBE.Windows. _
        CreateToolWindow(AddInInst, _
        "MyAddin.axdUserDoc", _
        "ActiveX Caption", guidMyTool, mobjDoc)
    mWin.Visible = True
End Sub
```

This code uses the Set statement to set the Window variable "mWin" to a reference to the window created by the function. This reference is then used to set the Visible property of the window to True.

6. Press CTRL+G to view the Immediate window.
7. Type "AddToIni" and press RETURN. You have just added the progID of the project to the VBAddIn.ini file, a necessary step when creating an Add-in.
8. Press F5 to run the project.
9. Minimize the Visual Basic window. This will prevent confusion as you will start another Visual Basic instance.
10. Start another instance of Visual Basic.
11. When the **New Project** dialog appears, press ENTER to start a new Standard .exe project.
12. On the **Add-Ins** menu, click **Add-In Manager** to display the **Add-In Manager** dialog box.
13. From the list of available add-ins, click **My Add-In**, the click **OK**. The ActiveX document you just created will now appear as a part of the Visual Basic development environment.
14. After viewing the document, you will want to shut it down. To do this, open the **Add-in Manager** dialog box again, and clear the **MyAddIn** checkbox, then click **OK**.

Adding a Second Window

Having an understanding of the mechanics of the function allows us to create a more complex scenario: creating a second window, containing a second ActiveX document, from the first.

To add a second ActiveX document

1. If you haven't already, quit the second instance of Visual Basic.

2. On the **Project** menu, click **Add UserDocument** and add a second UserDocument to the project.

3. Change the name of the UserDocument to "axdUserDoc2."

4. Add a TextBox control to the UserDocument.

5. Double-click the designer, and add the following code:

```
Public Property Get Text () As String
    Text = Text1.Text
End Property

Public Property Let Text(ByVal newText As String)
    Text1.Text = newText
End Property
```

   The preceding code creates a public property, which you will set when you create the window for the document.

6. In the **Project Explorer** window, double-click **AddIn** to open the code module. Add the following code to the Declarations section.

```
Public gAddIn As vbide.Addin
```

   The new line declares a global variable that will be set with a reference to the add-in instance.

7. In the **Project Explorer** window, double-click **Connect** to open the class module. Look at the OnConnect event, and modify it to resemble the following code:

```
Private Sub IDTExtensibility_OnConnection(ByVal _
VBInst As Object, ByVal ConnectMode As _
VBIDE.vbext_ConnectMode, ByVal AddInInst As _
VBIDE.AddIn, custom() As Variant)
    Set mWin = AddInInst.VBE.Windows. _
        CreateToolWindow(AddInInst, _
        "MyAddin.axdUserDoc", _
        "ActiveX Caption", guidMyTool, mobjDoc)
    mWin.Visible = True

    ' Add this new code:
    Set gAddIn = AddInInst
End Sub
```

   The new line sets the global variable to the add-in instance, making it available to create more windows.

8. In the **Project Explorer** Window, double-click **axdUserDocument** to bring its designer forward.

9. On the Toolbox, double-click the CommandButton icon to add a CommandButton control to the UserDocument.

10. Change the caption of the button to "Show Next."

11. Double-click the designer to open its code window, and add the following code:

```
Private mWin2 As Window
Private mDoc2 As axdUserDoc2
Const guidMyTool$ = "Xiang19X67Hangzhou4/27"

Private Sub Command1_Click()
    set mWin2 = gAddIn.VBE.Windows. _
        CreateToolWindow(gAddIn, _
        "MyAddin.axdUserDoc2", _
        "Second ActiveX Document", _
        guidMyTool, mDoc2)
    mWin2.Visible = True
    mDoc2.Text = "This is the second document."
End Sub
```

12. Press F5 to run the project.

13. Run another instance of Visual Basic.

14. When the **New Project** dialog appears, press ENTER to start a new Standard .exe project.

15. On the **Add-Ins** menu, click **Add-In Manager** to display the **Add-In Manager** dialog box.

16. From the list of available Add-ins, select the **My Add-In** checkbox, then click **OK**. The ActiveX document you just created will now appear as a part of the Visual Basic development environment.