Extend the Life of Your VB3 Apps

Click & Retrieve Source CODE!

Breathe some fresh air into those legacy applications by giving them access to OLE servers.

by Deborah Kurata

Not all Visual Basic developers are porting their VB3 applications to VB4. Maybe they are still concerned about finding bug-free replacement OCXs for those VBXs that have worked so tried and true. Maybe they just don't have a business need or available resources. Whatever the reason, deciding to keep your applications in VB3 doesn't mean you can't program with class.

As I discussed in my last Programming with Class column ["Creating OLE Servers," *VBPJ* March 1996], one of the exciting things you can do with classes is create an OLE server. OLE servers provide a mechanism for encapsulating business logic or other processing in an executable unit and allow other applications to access that processing through a set of predefined interfaces, called properties and methods. You can write OLE servers in either the 16-bit or 32-bit version of Visual Basic 4.0 (or in Visual C++).

So how does that help your Visual Basic 3.0 applications? Any VB3 application running on the same machine as a 16-bit or 32-bit OLE server can access that server through OLE Automation. A VB3 application can also access a 32-bit OLE server running in a 32-bit operating system through Remote Automation (see Figure 1). As a result, you can use the new features of Visual Basic 4.0 within Visual Basic 3.0 by encapsulating those features within a VB4 OLE server and accessing that server through OLE Automation.

CHANGING BUSINESS RULES

Picture this scenario: you come back from your vacation to find some major changes in your organization. The entire inventory control system must be rewritten to support new inventory control business rules. At this point, you have three options: (1) Return to your vacation and hope all will be fixed by the time you return, (2) Rewrite significant routines in all of the VB3 applications that access inventory, or (3) Develop an OLE server for the inventory control routines, remove a significant amount of code from the current VB3 applications that access inventory, and add code to these applications to call the new server.

Because neither of the first two options seem to further your career, the third option becomes your choice. To get started, follow the tips and samples in my last column to create an OLE server using VB4. If you've been wanting to learn VB4 in depth, this is a great opportunity. After that, modify the VB3 applications to become OLE client applications.

This is much easier than it sounds. Simply remove from the VB3 applications the code that is no longer required (because it is now in the OLE server), and replace it with the appropriate calls to properties and methods in the server application. I'll walk you through a prototype VB3 application that will access the OLE server described in the March column. You could adapt this code to work within any VB3 application.

It's always a good idea to start with a header comment block. Make sure the VB3 application has a clear header, and don't forget that all-important Option Explicit statement to ensure you don't misspell any variable names:

```
' Form Name:    frmInventory
' Author:       Deborah Kurata, InStep Technologies
' Date:         December 10, 1995
```

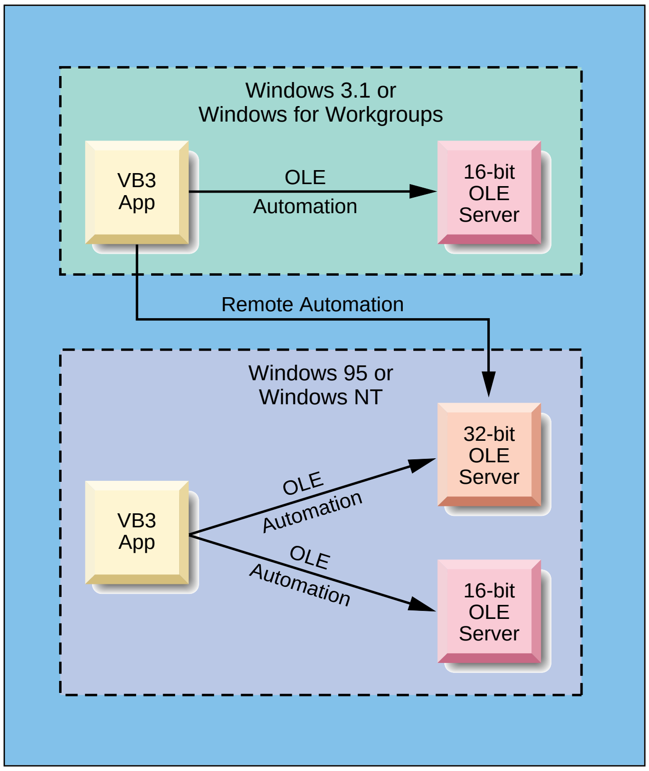


FIGURE 1 ***A New Lease on Life.*** *Extend the life of your VB3 app with OLE. Using OLE Automation or Remote Automation, your VB3 applications can access both 16-bit and 32-bit servers.*

Deborah Kurata is principal consultant and cofounder of InStep Technologies, a consulting group that designs and develops object-oriented Microsoft Windows applications. She is the author of Doing Objects in Microsoft Visual Basic 4.0, *published by Ziff-Davis Press, which focuses on a pragmatic approach to object-oriented design and development of Visual Basic applications. Reach her at InStep Technologies, 5424 Sunol Blvd. #10-229, Pleasanton, CA 94566; or find her leading the Beginner's Corner section of the* VBPJ *Forum on CompuServe at 72157,475.*

```
' Description:   Allows for adding and removing items
'                from inventory
' Revisions:
' 2/20/96        D Kurata  Replace code that processed
'                the inventory with code to access the
'                Inventory Control OLE server
Option Explicit
```

Next, create a member variable that will store the reference to the object from the top-level class of the server. This variable is called an object variable because it references an object:

```
' Private member variables
Dim m_Inv As object
```

Notice that in VB3 you declare the variable generically as "object," and not as the specific class of object as you would with VB4 (Dim m_Inv as CInventory). VB3 has no references and no knowledge of public classes registered by OLE servers. As a result, you cannot declare the object variable to be from a specific class. VB3 performs "late binding" as opposed to the "early binding" available with VB4: the VB3 application does not know about the class or its methods and properties until run time. No property and method syntax checking, no Object Browser support, and no "early binding" performance benefits are available in VB3.

Next write some code that defines constants for the command buttons on the form. Using constants instead of hard-coded numbers, known as "magic numbers," improves the readability and maintainability of the code:

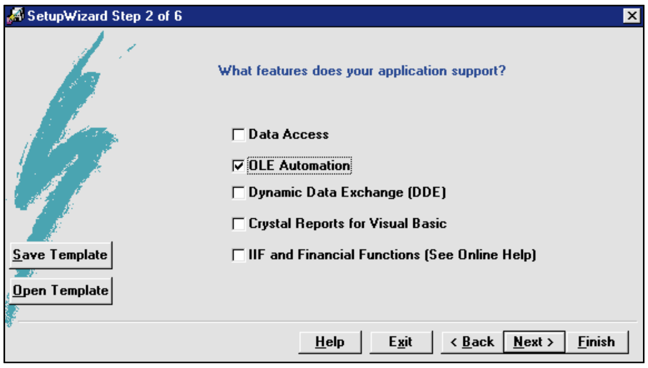
```
' Constants for the command buttons
Const iCLOSE = 0
Const iRECEIVE = 1
Const iSELL = 2
```

The top-level inventory object is created in the Form_Load event. The "Dim x as New CClass" and "Set x = New CClass" syntax used in VB4 is not available in VB3, but you can use the CreateObject function instead. The parameter to this function is the name of the OLE server application (Inventory) and the name of the class from which an object is to be created (CInventory):

```
Sub Form_Load ( )
Dim lCount As Long
On Error Resume Next

   ' Create an instance of the inventory class
   Set m_Inv = CreateObject("Inventory.CInventory")
   If Err <> 0 Then
      ' A more friendly error message would be
      ' displayed to the user here
      ' With a more detailed message printed to a log
      MsgBox Error$(Err)
      Unload Me
      GoTo EXIT_Form_Load
   End If

   ' Fill the combo box with values
   For lCount = 1 To m_Inv.Count
```


SetupWizard Step 2 of 6

What features does your application support?

- [ ] Data Access
- [x] OLE Automation
- [ ] Dynamic Data Exchange (DDE)
- [ ] Crystal Reports for Visual Basic
- [ ] IIF and Financial Functions (See Online Help)

FIGURE 2 ***Can't Do OLE Automation Without Them.*** *When you create the setup disks for VB3, include the files required for OLE Automation. If you use the VB3 SetupWizard, check the OLE Automation check box on the screen for Step 2.*

VB3

```
Private Sub cmdInventory_Click (Index As Integer)
Dim iTotal As Integer, ix As Integer
Dim sProductNumber As String

  ' Index into the collection
  ix = cboItems.ListIndex + 1

  Select Case Index

  Case iCLOSE
     Unload Me

  Case iRECEIVE
     ' NOTE: combo's are 0 based
     ' collections are 1 based
     sProductNumber = m_Inv.Item(ix).ProductNumber

     ' Add the defined amount to the inventory
     On Error Resume Next
     iTotal = m_Inv.AddInventory(sProductNumber, _
        Val(txtReceived.Text))
     If Err <> 0 Then
        ' A more friendly error message would be
        ' displayed to the user here
        ' With a detailed message printed to a log
        MsgBox Error$(Err)
     End If

     ' Display the revised inventory amount
     txtTotal.Text = iTotal

  Case iSELL
     ' Product number for the inventory item
     sProductNumber = m_Inv.Item(ix).ProductNumber

     ' Remove defined amount from the inventory
     On Error Resume Next
     iTotal=m_Inv.RemoveInventory(sProductNumber, _
        Val(txtReceived.Text))
     If Err <> 0 Then
        ' A more friendly error message would be
        ' displayed to the user here
        ' With a detailed message printed to a log
        MsgBox Error$(Err)
     End If

     ' Display revised inventory amount on the screen
     txtTotal.Text = iTotal
  End Select
End Sub
```

LISTING 1 ***It's Just a Click Away (from VB4).*** *Clicking on a button in a VB3 application does the processing in a VB4 application. Put this code in the Button_Click event for the button control array.*

```
      cboItems.AddItem _
         m_Inv.Item(lCount)._
         ProductNumber & _
         Space(5) &  m_Inv._
         Item(lCount).ProductName
   Next lCount
EXIT_Form_Load:
End Sub
```

If the object was created without an error, the code accesses the server to get the Count property, then loops through all inventory items and loads the items into a combo box control. Use the Item method of the CInventory class to retrieve the item and the ProductNumber and ProductName properties of the Item to retrieve the product information.

Because the user interface of this test OLE client application provides three buttons—close, receive inventory, and sell inventory—you also need to write code in the Button_Click event for the button control array (see Listing 1). This code demonstrates how you can use the methods and properties of the OLE server within a VB3 application.

As you do with VB4, set the object to Nothing when the object is no longer necessary:

```
Private Sub Form_Unload (Cancel As _
   Integer)
   ' Clear the instance
   Set m_Inv = Nothing
End Sub
```

USING JET 3.0 AND RDO FROM VB3

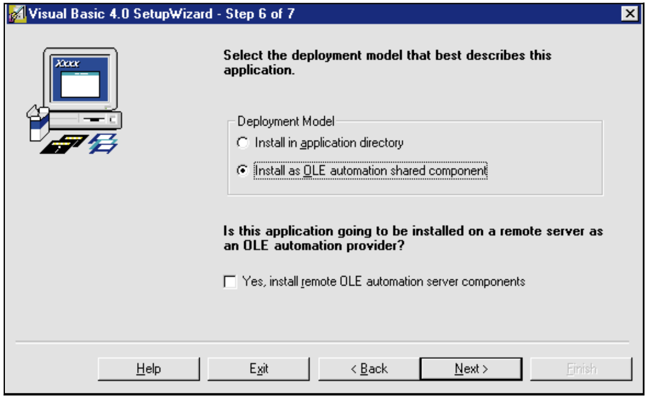
Not only can you revise your VB3 apps to accommodate business rules that have changed, but you can also give them new functionality. Through OLE servers, your VB3 applications can access a lot of features that you might have thought were available only to VB4 applications. For instance, although there's no "compatibility layer" to let your VB3 apps access the cool new features of Jet 3.0, you can create an OLE server to do the job. (For details on Jet's new features, see "Moving to Jet 3.0" in the December 1995 issue of *VBPJ.*)

This OLE server will act as a database server. All requests to the database could be submitted to this OLE server and all requests from the database would pass back through the OLE server. Suddenly, all the new performance optimizations, recordset processing, and replication capabilities are available to your VB3 application.

You face one restriction, however, in this scenario. Because Jet 3.0 is 32-bit only, the OLE server needs to be 32-bit and needs to be executed on a 32-bit operating system, such as Windows 95 or Windows NT. To access the 32-bit OLE server, your VB3 application either needs to run on that platform also, or use Remote Automation from a machine running Windows 3.1 or Windows for Workgroups (see Figure 1).

If you are accessing remote databases from your VB3 application, you could improve your performance by using the new remote data objects provided in the Enterprise Edition of Visual Basic 4.0. These remote data objects provide a thin layer over ODBC for significant performance improvements. For details about RDO, see "A Walking Tour of RDO" in the March 1996 issue of *VBPJ.*

Use RDO from your VB3 application the


Visual Basic 4.0 SetupWizard - Step 6 of 7

Select the deployment model that best describes this application.

Deployment Model
- ( ) Install in application directory
- (●) Install as OLE automation shared component

Is this application going to be installed on a remote server as an OLE automation provider?

☐ Yes, install remote OLE automation server components

FIGURE 3 ***Be Sure to Share.*** *When creating the setup disks for the OLE server, be sure to identify it as a shared component. When the OLE server is installed, it will automatically be registered for use.*

same way you would access Jet 3.0. Simply encapsulate the desired processing in a 32-bit OLE server. Then modify the VB3 application to access the server through OLE Automation or Remote Automation.

In addition to creating OLE servers to use RDO and Jet 3.0, you can create an OLE server to facilitate deployment of your application in different languages. There is no easy way to localize VB3 applications. Your only choices are to store all the strings in a database and maintain the database, add a lot of If…Else…End If or Case statements in your code, or develop multiple versions of the application.

With VB4, you can use resource files to store and retrieve needed strings. (You can also store bitmaps and other binary data in the resource files.) To make use of this functionality in your VB3 application, create a localization OLE server that provides strings or other resources from the resource file to your VB3 application.

In closing, I should say a few words about installation issues to ensure that your use of VB4 OLE servers from VB3 is successful. First, there is no easy way to install the VB4 OLE server with your VB3 application. The easiest thing to do is provide two sets of install disks: one for the VB3 application and one for the VB4 application.

Second, when creating the setup disks for VB3, be sure to include the files required for OLE Automation. If you use the VB3 SetupWizard, check the OLE Automation check box on the screen for Step 2 (see Figure 2). VB3 cannot use OLE Automation without these files. Third, when creating the setup disks for the OLE server, identify it as a shared component. When the OLE server is installed, it will automatically be registered for use (see Figure 3).

So before you groan at your boss' announcement that the company won't be porting its VB3 applications, consider extending their life by turning them into OLE client applications. This way they can tap into the capabilities of VB4 by accessing VB4 OLE servers that provide whatever functionality you need. ☒