Chapter 6
Creating and Managing COM Components

Microsoft's Component Object Model (COM) is one of those technologies that simply makes sense. It is a natural fit for object-oriented development and n-tier design. COM allows you to make your VFP objects available to other software and programming environments, thus making a flexible technology for business components in the middle tier. We will show how to build COM components, the different types of COM components available, how to register them, and how to manage them with the Component Gallery. As well, we will touch on error-handling issues with COM and special considerations when using COM and Microsoft Transaction Server.

Evan is a fan of the BBC series "Junkyard Wars." Two teams of four people are asked to build a device to perform a specific task. They have about 10 hours, using only items gathered from a scrap yard. The teams only know the simplest of information about what they have to accomplish. The host of the show may ask them to "build a flying machine" or "build a device to raise a car submerged in water." Evan's favorite episode had the two teams building a device to knock down a wall. One team built an A-frame with a pole suspended by chains. One end of the pole had a heavy weight on it, and the team members pulled the pole back and forth using muscle power. The second team built a hydraulic arm with a jaw at the end. To make a long story short, the team with the hydraulic arm lost because their complicated system broke down and because one of the walls had a roof and they were unable to grip it with the jaw. The team with the pole was very slow, but because of the simplicity of the machine, it didn't break down. The pole was versatile as well; they could shorten or lengthen the chains to hit the wall at any point.

With few exceptions, teams win these contests because of the simplicity and versatility of their designs. Simplicity frees the designers to focus on solving the problem rather than on the details of implementing the solution. Versatility allows unexpected situations to be accommodated with fewer system changes. The Component Object Model (COM) and Distributed COM (DCOM) provide Visual FoxPro developers with both simplicity and versatility in deploying applications. Let's look at what they offer us toward making our applications simple and versatile.

What is COM?

COM is a technology that allows a developer to make objects written in a variety of programming languages available to other programming languages and software. COM allows us to deploy an application on a single client computer or across several servers via DCOM.

Simplicity comes from the way COM is implemented. Take a Visual FoxPro class definition and simply add the OLEPUBLIC keyword, and compile the classes into an .EXE or .DLL. Then, you can instantiate the class in any of the following ways:

Pure VFP: `oJoe = CREATEOBJECT("Staff")`
COM: `oJoe = CREATEOBJECT("ComName.Staff")`
DCOM: `oJoe = CREATEOBJECT("DcomName.Staff","MyServer.com")`

Once an object has been instantiated, it is used in the same way regardless of whether VFP, COM, or DCOM has been used. What could be simpler?

COM offers a variety of choices in how it is deployed. We can host applications locally on a single machine. If necessary, we can switch to DCOM when an application needs to be shared by more than one user via a server.

Because COM is used for implementing objects, it is a natural fit for object-oriented programming (OOP) used in Visual FoxPro.

Creating COM components

Because COM components can be deployed locally and then scaled up to remote servers, they are ideal for encapsulating business rules in the business tier. As long as all software uses the COM business components to access data, rather than accessing the data directly, you can be sure that your business rules are being respected. The business rules are easy to spot in **Listing 1**, and because they are centralized in the component, they are easy to change should the business rules change.

***Listing 1**. COM component showing business rules for adding patients.*

```
DEFINE CLASS Patient AS SESSION OLEPUBLIC
  * Adds new patient to database
  * Returns .T. if successful
  * Returns .F. if a business rule is violated

  DATASESSION = 2    && private datasession

  PROCEDURE Init
    USE Patients SHARED in 0
  ENDPROC

  PROCEDURE Destroy
    USE IN Patients
  ENDPROC

  PROCEDURE AddData
    LARAMETERS pcName, pnAge

    * Patient name must not be blank
    IF ALLTRIM(pcName) == ""
      RETURN .F.
    ENDIF

    * Patient must be 18 or over
    IF pnAge < 18
      RETURN .F.
    ENDIF

    * No business rule violated, add data to database
    SELECT Patients
```

```
    INSERT INTO Patients (FullName, Age) VALUES (pcName, pnAge)
    RETURN .T.
  ENDPROC

ENDDEFINE
```

COM server types

COM components can be created as in-process or out-of-process. In-process components can be single-threaded or multi-threaded. At compile time, you choose which type of component to build: out-of-process .EXE, single-threaded in-process .DLL, or multi-threaded in-process .DLL.

An in-process component is a .DLL that runs in the same address space as the program that calls it. In-process components run faster because there is no inter-process communication. The downside is that a crash in the .DLL can crash the calling client. In addition, to update a .DLL that is part of a Web application, you must shut down the Web server. The only way to run an in-process component on a remote computer is via MTS. A multi-threaded .DLL can increase responsiveness of the application, especially if there are long processes on the system, because the shorter processes will be able to interrupt the long process and give the users the impression that the system has a better response time. Multi-threading comes at the cost of overhead of managing the processes.

An out-of-process component is an .EXE that runs in a separate process from the calling client. Though slower than a .DLL (as mentioned previously), a crash on the server won't bring down the client. An .EXE can be updated without stopping the Web server. The out-of-process component can run on a remote computer via DCOM.

Business rules

Business rules are the constraints of the system that you are modeling. Examples of business rules are:

- Every invoice must have at least one product on it.
- All patients must be 18 years or older.
- Customers paying by check must provide three pieces of identification.

The component is then compiled into a COM .DLL called BIZCOM.DLL. **Listing 2** shows an example of its use in VFP. Put all of the components and tables into one directory and SET DEFAULT TO that directory. Evan uses session-based classes rather than custom-based because it allows him to set private data sessions. This means that for non-COM-based applications, the data manipulations in the objects won't affect each other. Also, it means that he can use these class definitions in a pure VFP environment or a COM environment.

***Listing 2**. Using a COM component with good and bad data.*

```
* Instantiate the CLASS as an object
oPatient = CREATEOBJECT("BizCom.Patient")

* Add a patient
IF oPatient.AddData("Humphrey Bogart", 82) = .T.
  MESSAGEBOX("Data saved")
ELSE
   MESSAGEBOX("Data not saved")
ENDIF
* Attempt to add another patient that breaks the rules
IF oPatient.AddData("Orphan Annie", 5) = .T.
  MESSAGEBOX("Data saved")
ELSE
  MESSAGEBOX("Data not saved")
ENDIF

* Release the object
RELEASE oPatient
```

Interacting with other components

COM components can call each other to delegate responsibility from one to another. **Listing 3** shows the oEmployee object delegating the lookup of the department name to the oDept object. **Listing 4** shows how the method is called in code. We purposefully haven't listed the code for the oDept object because we don't care how it is implemented. All we know is that we pass it an employee primary ID and it sends back the department name. The object could be getting the data out of a VFP table on the C:\ drive, or from a SQL Server table half a world away!

***Listing 3**. The oEmployee object delegates department name lookup to the oDept object.*

```
DEFINE CLASS Employee AS SESSION OLEPUBLIC
  * Employee Biz object
  DATASESSION = 2  && private datasession

  PROCEDURE Init
    USE Employee SHARED IN 0
  ENDPROC

  PROCEDURE Destroy
    USE IN Employee
  ENDPROC

  PROCEDURE GetDeptName
    LPARAMETERS pnEmployeePK
    LOCAL lcDeptName

    * Get the department foreign key from employee table
    * Returns department name if found
    IF NOT USED("Employee")
      USE Employee IN 0
    ENDIF
```

```
    IF SEEK(pnEmployeePK, "Employee", "pkId")
      * Delegate retrieving the department name to oDept object
      oDept = CREATEOBJECT("Dept")
      lcDeptName = oDept.GetName(Employee.fkDept)
      RELEASE oDept
      RETURN lcDeptName
    ELSE
      RETURN "Employee not found"
    ENDIF
  ENDPROC
ENDDEFINE
```

***Listing 4**. Using the GetDeptName() method of the oEmployee object.*

```
* Instantiate the employee business class
oEmployee = CREATEOBJECT("BizCom.Employee")

* Get the dept names for employees with primary keys 1, 2, 4, 5
? oEmployee.GetDeptName(1)
? oEmployee.GetDeptName(2)
? oEmployee.GetDeptName(4)
? oEmployee.GetDeptName(5)

* Release the employee object from memory
RELEASE oEmployee
```

COM error handling

Error handling with COM components is especially important because often the components have no user interface and they can be running on computers with no human operator. In a conventional application, if your program can't determine what to do in the case of a particular error, you can display the options to the user. This may be something like the classic "The staff data cannot be accessed, would you like to Retry, Ignore, or Cancel?" The user can try one or more of the options to see whether anything works before calling tech support or rebooting. A COM component can only report the error back to the calling component and log the error. If a message were to display on a COM .EXE component running on a server, it could lock the component because an error message would pop on the server's monitor. Without anyone there to press a Continue or OK button, the component would be stuck and couldn't respond to requests. You can prevent a component from having a wait state (such as an error message from an unhandled error) by using SYS(2335). COM .DLL components cannot and do not natively have any UI capabilities, so SYS(2335) is unnecessary.

A good error-handling strategy anticipates problems and handles them gracefully. It's always a good idea to check that all the parameters have been passed in and that they are of the expected type. Instead of simply opening a table, check whether it is already open with the USED() function. When checking the results returned from a function or object, test for all return values, even "impossible" ones. Though we are discussing COM error handling, this advice applies to error handling for all types of applications.

For errors that can't be anticipated, a good strategy will return as much information as possible and bow out gracefully. Every class definition should have code in its Error() method,

which fires automatically when an error occurs. This differs from a conventional application, where you have to invoke an error handler with the ON ERROR command. When the object's Error event fires, the Error() method should try to handle all the errors that are specific to the class. Any that it can't handle can be passed up the class hierarchy. If the parent class can't handle the error, an error-handling class can be called.

The function COMRETURNERROR() lets our COM components raise errors. The beauty of this mechanism is that the client tier doesn't have to have to handle COM errors differently than errors that occur inside of the client tier itself. The error information is gathered using AERROR(). **Listing 5** shows how to use COMRETURNERROR() and also demonstrates one method of logging the error to a file. In Listing 5, we're trying to keep the example simple; in a real component, you want to try to handle the error locally if you can.

***Listing 5**. Component showing COM error handling and logging.*

```
DEFINE CLASS Invoice AS SESSION OLEPUBLIC
  * Invoice Biz object
  * Demonstrates COM error handling
    PROCEDURE DoSomething(pnDeptPK)
    * generate a fake error using the error command
    ERROR "A really terrible error"
  ENDPROC

  PROCEDURE Error(pnErrorNum, pcMethodName, pnLineNum)
    LOCAL lcErrorMess

  * To keep example simple, don't handle error, pass it back to
  * calling application
    lcErrorMess = ;
      DTOC(DATETIME())+ ;
      " | Error Number:" + TRANSFORM(pnErrorNum) + ;
      " | Method Name:" + pcMethodName + ;
      " | Line Number:" + TRANSFORM(pnLineNum) + ;
      " | Message:" + message()
    * Append to error log text file
    STRTOFILE(lcErrorMess + chr(13), "error.log", .t. )
    * Raise a COM error to notify client
    * The client will handle the error as instructed
    COMRETURNERROR("MyServer",lcErrorMess)
  ENDPROC
ENDDEFINE
```

Listing 6 shows that, in the client code, COM errors are handled exactly the same as other VFP errors. In this code fragment, we simply display the error information, though in a real application we would probably choose to handle the error more gracefully.

***Listing 6**. COM error handling at the client.*

```
ON ERROR ErrorHandler()  && Enable error handler

* Instantiate invoice class to an object
oInvoice = CREATEOBJECT("BizCom.Invoice")
```

```
nResult = oInvoice.DoSomething(1)

PROCEDURE ErrorHandler()
  DIMENSION laError[1]

  * Load error info into an array
  * In a real application we would try to handle the error
  AERROR(laError)
  LcErrorMessage = ""
  FOR i = 1 TO 7
    LcErrorMessage = lcErrorMessage + laError(i) + CHR(13)
  ENDFOR

  MESSAGEBOX(lcErrorMessage)
ENDPROC
```

Designing components for MTS

Microsoft Transaction Server (MTS) is a run-time environment for COM components that allows caching of the components to improve performance. For background on what MTS is and why you would want to use it, see Randy Brown's article "Microsoft Transaction Server for Visual FoxPro Developers" in the MSDN Library, as listed in the "Further reading" section at the end of this chapter.

When designing COM components to be used with MTS, we need to consider the following:

- Stateless components that don't rely on state being saved are more efficient in MTS.
- COM components used in MTS must be in-process .DLLs.
- SetComplete and SetAbort methods let MTS know when you won't need the component for a while.

Stateless components are more efficient for MTS because the context of the component doesn't need to be restored each time the component is called. This means less work for the server, meaning that the server can handle more requests and scale better. As our tech editor spotted, the code samples earlier in the chapter are statefull because they rely on the data tables being opened in the Init() method of the object. For a discussion of state and statelessness, see **http://msdn.microsoft.com/library/techart/d3dalchg.htm**.

Only in-process COM .DLLs run in MTS. Simple, out-of-process .EXEs simply don't run in MTS, so recompile your components into .DLLs.

Even if we aren't using transactions, SetComplete and SetAbort tell MTS that the object we are using can become stateless. This allows MTS to deactivate the object and free up resources on the server. At the end of each object method, issue SetComplete before the RETURN statement.

Transaction support for COM components in MTS

MTS supports database transactions via Distributed Transaction Coordinator (DTC). Both Microsoft SQL Server and Oracle are supported, but unfortunately Visual FoxPro database transactions are not. With Visual FoxPro tables, the objects must manage the transactions. DTC

means that transactions are handled by MTS and that transactions can be committed or rolled back over heterogeneous systems databases and servers. Before DTC, the developer had to manage these potentially complex transactions himself.

To begin a transaction, call GetObjectContext(.T.). Then, do the database operations. Finally, call SetComplete() to commit the transaction or SetAbort() to revert it.

Managing and deploying COM components

Now that we have talked about developing COM components, let's look behind the scenes at building and managing the components. We'll look at using the Component Gallery to manage components, COM server types, registering components, and creating and management packages with MTS Explorer.

Using the Component Gallery

The Component Gallery is accessed from the Tools | Component Gallery menu option. The Gallery is an interface for grouping components into categories called "catalogs." Catalogs are pointers or shortcuts to the actual components, and they can be further broken down into folders. They offer us a convenient way to organize not only components, but other classes, projects, and really just about anything. Each catalog is a .DBF file. The Component Gallery is just another tool for helping to organize VFP development. It is pan-project and goes beyond just .VCX-based classes like the Class Browser.

To add a new catalog:

1. Click the Options button on the toolbar.
2. Click the Catalogs tab (see **Figure 1**).
3. Click the New… button (see **Figure 2**).
4. Type a catalog name and location.
5. Click the Save button.
6. Click OK to leave the Options dialog.

To add a new folder:

1. Choose (in the left pane) the catalog to which you want to add a folder.
2. Right-click on the right pane of the Component Gallery.
3. Choose New Item, then Folder.
4. Right-click on the folder called "New Folder," and then click Properties.
5. Enter a name and description, and then click the OK button.

To add a component to a catalog:

1. Choose the catalog or folder to add to.
2. Right-click on the right pane.

3. Select New Item, then the type of component (Activex, Class, Datasource, File, Form, Image, Menu, Program, Project, Report, Sample, Web Site).
4. Browse to find the component, and then click the OK button.
5. Right-click the component, and choose Properties to add a description and icon.

You can set various options in the Component Gallery by clicking the Options button on the toolbar. A handy feature is that you can drag components into your VFP projects.

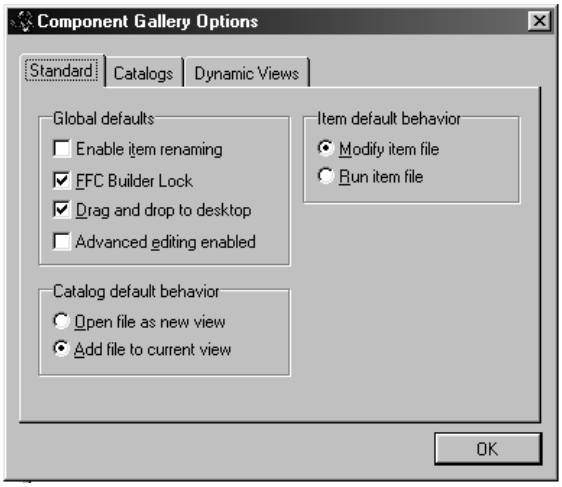

***Figure 1**. The Component Gallery Options Standard tab. You can choose the default behavior when a component is clicked.*

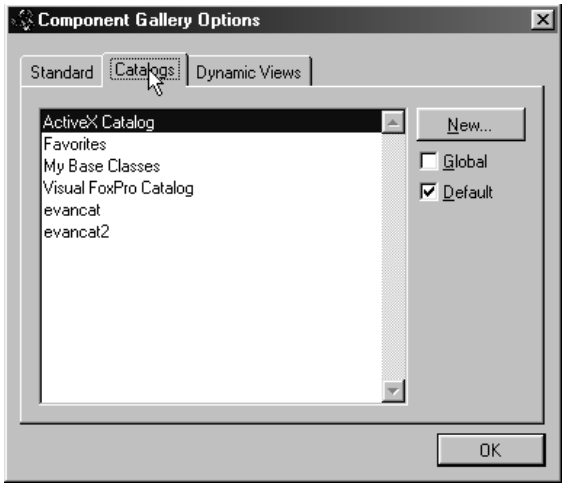

***Figure 2**. The Component Gallery Options Catalogs tab. Evan has added his own catalogs.*

Registering and unregistering COM components

COM components must be registered with Windows (that is, in the Registry) before you can use them. For .EXE components:

- Register by using **`MyExe /regserver`**
- Unregister with **`MyExe /unregserver`**

For .DLL components:

- Register with **`REGSVR32 MyDll.DLL`**
- Unregister with **`REGSVR32 /u MyDll.DLL`**

Creating packages and managing components with MTS Explorer

MTS lets you browse the components registered on a machine. See Chapter 10, "Deploying an Application," for more details. Also, see the Microsoft white paper, "Microsoft Management Console: Overview," which is referenced in the "Further reading" section of this chapter.

Following are some common things that are done with MTS Explorer (also called Microsoft Management Console). **Figure 3** shows the layout of Microsoft Management Console, a familiar interface to those who are accustomed to using to Windows Explorer.

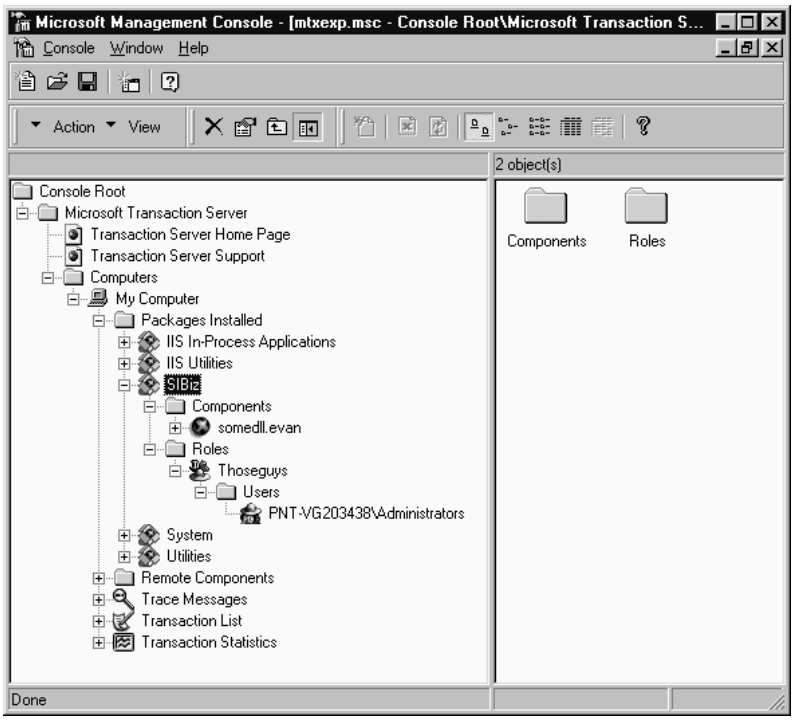

***Figure 3**. Microsoft Management Console showing the SIBiz package.*

To create an empty package:

1. From the command line, run MMC (Microsoft Management Console).
2. Navigate down the tree in the left pane to “Packages Installed” (see Figure 3).
3. On the toolbar, choose Create A New Object.
4. Choose Create An Empty Package.
5. Type in a name for the package, and then choose Next.
6. Select the users that can use this package, and then choose Finish.

To add a component to a package:

1. In the left pane, select the package.
2. Select the component folder under the package.
3. Use Windows Explorer to find your component.
4. Drag and drop it into the right pane.

Using role-based security

MTS allows you to leverage Windows NT and Windows 2000’s security features using role-based security. A role is a group of users that can access an MTS component.

To add a role to a package:

1. Under the package in the left pane, select the subfolder Roles.
2. Right-click the subfolder and select New, then Roles.
3. Type in a name for the role, then click OK.
4. In the left pane, select your role, and then select the subfolder Users.
5. Right-click and select New, then User.
6. Select each user, and then click Add. Do this for each user you need to add to the group.
7. Click OK.

Sample Question

You need to register a component called TheDLL.DLL. Which syntax will do the job?

A. TheDLL.DLL /R
B. TheDLL.DLL /Reg
C. REGSVR32 /R TheDLL.DLL
D. REGSVR32 TheDLL.DLL

Answer: D

Further reading

- *Internet Applications with Visual FoxPro 6.0*, Rick Strahl, Chapter 9, "Visual FoxPro and COM"
- "Error Handling in Visual FoxPro," Doug Hennig, includes article and extensive code for handling errors during execution, **www.stonefield.com/pub/errorh.zip**
- **http://fox.wikis.com/wc.dll?Wiki~ComErrorHandling**
- **http://fox.wikis.com/wc.dll?Wiki~COMComponentExample**
- **http://fox.wikis.com/wc.dll?Wiki~ComReturnError**
- **http://fox.wikis.com/wc.dll?Wiki~ComErrors**
- **http://fox.wikis.com/wc.dll?Wiki~ErrorEventStrategy**
- "Microsoft Management Console: Overview," white paper, **http://microsoft.com/windows2000/library/howitworks/management/mmcover.asp**
- "Microsoft Transaction Server for Visual FoxPro Developers," Randy Brown, Microsoft Corporation, **http://msdn.microsoft.com/library/default.asp?URL=/library/techart/mtsvfp.htm**
- "Using Microsoft Transaction Server With VFP," Rick Strahl, **www.west-wind.com/presentations/mts/mts.htm**
- **http://fox.wikis.com/wc.dll?Wiki~SettingUpMTS**
- "How Microsoft Transaction Server Changes the COM Programming Model," David Chappell, **http://msdn.microsoft.com/library/periodic/period98/mtsjan.htm**
- **http://fox.wikis.com/wc.dll?Wiki~UsingAndDevelopingForMTS**
- "The Microsoft Component Gallery," Steven M. Black, **http://msdn.microsoft.com/library/default.asp?URL=/library/techart/Vfpgallery.htm**
- *Advanced Object Oriented Programming with Visual FoxPro 6.0*, Markus Egger, Chapter 5, "OOP Standards"
- *The Fundamentals: Building Visual Studio Applications on a Visual FoxPro 6.0 Foundation*, Whil Hentzen, Chapter 17, "The Component Gallery"
- *Hacker's Guide to Visual FoxPro 6.0*, Tamar Granor and Ted Roche, "Hacking the Class Browser and the Component Gallery"