Simple OLE Data Server

by Scott Whittlesey 1/5/96

Purpose

This project was created as a learning exercise in creating OLE servers in VB4. It is submitted in hopes that it may be of some use to others who are also learning this aspect of the language. The source code should be examined in addition to the comments in this document.

The project includes elements of the following concepts:

- Use of an OLE server as a data server with asyncronous processing.
- Out-Of-Process OLE Servers (also referred to as 'cross-process').
- Maintaining an Instance Count.
- Controlling Instantiation
- Class Modules.
- Conditional Compilation.
- Collections.
- Raising errors.

Requirements

- Windows 95
- VB4 Professional Edition/32-bit

Project Overview

In this project, an OLE Server application opens a database and processes requests for data from one or more Client applications running on the same machine. Both VB projects are included: OLESRV3.VBP (the Server) and CLIENT3.VBP (the Client). This approach could be used in situations where data from a single source is to be supplied to several client applications; it eliminates the need for each Client to open its own database connection and contain code to retrieve the data.

Sequence of events:

- In the Client, a command button executes a function which passes a request, via OLE, to the Server. Code in the Client continues to execute while the Server is dealing with the request.
- The Server stores the request in a queue.
- The Server continuously monitors the queue using a Timer control.
- When a request is found in the queue, the Server retrieves a value from a table in the database, returns the value to the Client via OLE, then removes the request from the queue.
- When the Client receives the value, it displays it in a status label; processing (in this case, a timer and a text box) is not noticeably interrupted.

The Server has a visible form for the sole purpose of displaying status messages: as each request is received from a Client, a message is displayed. Another message is displayed as each request is processed.

Each Client creates an instance of class DataServer exposed by the Server. Only one instance of the Server is run; it provides each Client with a reference to a DataServer object.

Each Client also creates object **objRequest**, used for communicating with the Server. On startup, the Client application is assigned a unique ID, which is also used as an ID property for **objRequest**.

When the first instance of the Client is run, the Server is started automatically. This is a characteristic of OLE servers in VB4. Several Clients can be run. When the last Client application ends, the Server closes its database and ends.

The Server will accept only one request per Client at a given time. An error occurs (and is handled) if the Client attempts to send more than one request to the Server.

Suggested procedures for running the project:

Testing with a single Client:
Open both projects in separate instances of VB4. Run the Server first, then load the Client, select *Test simple out of process server* from the Tools/References dialog, and run the Client. Breakpoints and other debugging can be performed in either or both projects.

Testing with multiple Clients:
- Run the Server in the VB environment.
- Compile the Client and close the Client's instance of VB.
- Run two or more instances of the Client. Note: Please wait at least 1 second before starting each instance of the Client. VB's 'Now' function is used to create a unique ID for each instance, and is only accurate to the second; if more than one instance of the Client is started in the same second, the ID's will not be unique. In an actual project, it would probably be a good idea to use some other means to create an ID, such as a user-login ID or the **TimeGetTimer()** API function.

Server Project: OLESRV3.VBP

Modules:
- SRVTIMR.FRM (frmTimer)
- DATSERVR.CLS (CDataServer)
- OLESRV3.BAS (Module1)

Settings in /tools/options:
project:
- Startup form: Sub Main
- Project Name: ServerTest3
- Startmode: OLE Server
- Application Description: Test simple out-of process server.

advanced:
- Error Trapping: Break in Class Module (required for proper error handling)
- Conditional Compilation Arguments: Testing = -1 (see comments in code)

OLESRV3.BAS (Module1)
Declarations
- Declares variable **mdb** As Database; private to Module1.
- Declares public integer **InstanceCount**. It is public so that it is visible to instances of DATSERVR.CLS. Whenever an instance of this class is created, **InstanceCount** will be incremented (in the **Initialize** event). Whenever an instance is destroyed, **InstanceCount** will be decremented (in the **Terminate** event). Please refer to the description of DATSERVR.CLS for more detail.
- Conditional compilation statements:

```
#If Testing Then
        Const dbPath = ""
#Else
        Const dbPath = ""
#End If
```

The variable **Testing** is set from the VB4 Tools/Options/Advanced dialog. If **Testing** is set to -1 (values in the Conditional Compilation Arguments text box must integers), constant **dbPath** is set to the path in the first condition above; otherwise App.Path will be used (see **Sub DBOpen**). **dbPath** can be set to any programmer-defined path or expression. This functionality is not required for this project and has been included merely as an example.

Sub Main
This checks the StartMode property of the App object to make sure the program was started by another application

requesting a reference to an object (**vbSModeAutomation**). If the program was started manually by running the .exe (**vbSModeStandalone**), a message is displayed and the program ends. This can be tested by running OLESRV3.EXE from the Explorer or File Manager. Note: in many situations it is normal for an out-of-process OLE server to be started by running the .exe, but as one of the purposes of this project was to demonstrate the automatic loading and terminating of an OLE server program, it will not run as a standalone.

Sub DBOpen, Sub DBClose
Opens SERVER2.MDB when the first instance of **CDataServer** is created; closes the database when the last instance of **CDataServer** is destroyed. Please refer to the description of DATSERVR.CLS for more details.

Function NxtItm()
This function provides the database functionality for the project. It creates a recordset of type dynaset (note the syntax change from VB3), retrieves the "next available" number from the database, increments it, and writes it back. The function returns the value that was retrieved from the database, before being incremented.

This function is called from **Sub Timer1_Timer** in **frmTimer**.

SRVTIMR.FRM (frmTimer)

The primary purpose of this form is as a container for a Timer control which continuously checks the "queue" (public collection variable **gcolQueue**) to determine whether there are any requests to process.

The secondary purpose is to display status messages for actions being performed by the program. Messages are displayed in a standard ListBox, which can be cleared by clicking on the **Clear** button.

In an actual situation it is likely that this form would remain hidden, though out-of-process OLE servers used for different purposes could very well have visible forms.

Sub Timer1_Timer:

```
Sub Timer1_Timer
Dim lngCount As Long

If gcolQueue.Count > 0 Then
  timer1.Enabled = False
  List1.AddItem gcolQueue(1).ID & ": Processing request "
  List1.Refresh
  lngCount = NxtItm()
  gcolQueue(1).Notify (lngCount)
  gcolQueue.Remove 1
  timer1.Enabled = True
End If
```

When the timer event fires, the program checks the Count property of the collection. This is a built-in property (as a matter of fact, the only property) of collection objects.

The collection is made up of references to objects. In this project, these objects are of class **CRequest** as defined in the client project CLIENT3.VBP. However, any object can be passed, as long as it has a notify event and a unique ID (please refer to the discussion of **CDataServer** for more details).

It is important to understand that an object exists in the application that contains its class module, and has access to public variables and procedures in that application. In this case the Request objects exist in the Client applications; collection **gcolQueue** in the Server contains only references to those objects. Most documentation will refer to "objects" in a general way, without making the distinction between "object" and "object reference", so it is helpful to take the time to visualize where the object actually resides.

If the collection contains at least one object, **Function NxtItm()** is called to retrieve a value from the database.

The **Notify** event of the object being processed is called, passing the retrieved value as a parameter. (The first object in the queue is always the one to be processed).

Keeping in mind that the actual Request object exists in the client application: when the **Notify** event fires (**Public Sub Notify()** in module REQUEST.CLS), the returned value is used to set a label caption on a form in the client application. In actual use, the **Notify** event might set a global variable and/or execute additional statements.

Once the value has been passed back to the client via the **Notify** event, the object reference is removed from the collection with the statement: **gcolQueue.Remove 1**. This removes the reference to the Request object from the collection in the Server, but the object itself still exists in the client application.

There are only two additional things that happen in this procedure: the timer shuts itself off temporarily while processing a request, and a message is added to the listbox showing the ID of the object reference being processed.

DATSERVR.CLS (CDataServer)

Settings:

- **Instancing: 2 - Creatable MultiUse**. For another program to be able to create an instance of a class, Instancing must be set to either *1 - Creatable SingleUse* or *2 - Creatable MultiUse*. *SingleUse* is not often used; it would result in a separate instance of the entire .exe being run each time a DataServer object was created. *MultiUse* allows a single .exe to create many instances of its public classes.
- **Public: True.** This is necessary for the class to be visible outside of the current project.

CDataServer is a very simple class - it has no properties and only one method. It serves the following purposes:

- To control the opening and closing of the database, in the Initialize and Terminate events respectively.
- To load and unload the project's form (see srvtimr.frm, above).
- To register a request, in the form of an object reference, from the client application and add the reference to the queue.

Class Initialize:

```
Private Sub Class_Initialize()
If InstanceCount = 0 Then
    frmTimer.Show
    Call DBOpen
End If
InstanceCount = InstanceCount + 1
End Sub
```

InstanceCount is public (global to the project). The first time a Client creates an instance of **CDataServer**, the **Initialize** event loads and shows the Server's form and calls a procedure in Module1 to open the database.

Class Terminate:

```
Private Sub Class_Terminate()
InstanceCount = InstanceCount - 1

If InstanceCount = 0 Then

  '-- Make sure collection is empty; any
  '   existing requests will be terminated.
  Do While gcolQueue.Count > 0
    gcolQueue.Remove 1
  Loop

  '-- Close the database
  Call DBClose

  '-- Get rid of the timer form
  Unload frmTimer
```

```
  Set frmTimer = Nothing

  '-- The application should now shut itself down

End If

End Sub
```

This event will fire each time a client application destroys its instance of **CDataServer**, which occurs when the client application is shut down. **InstanceCount** is decremented so that actions can be taken when no more instances of **CDataServer** exist. In this project, the actions are:

- Remove any unprocessed requests in the collection; this situation "should not happen" but the code is included as a safety mechanism.
- Close the database.
- Unload the form and set it to Nothing.

At this point, all the conditions required for an OLE server application to automatically end are met (see the Professional Features manual, Creating OLE Servers, p82):

- There are no external references to your objects.
- Your OLE Server has no forms loaded.
- There is no code in your OLE server currently executing, and none in the call list waiting to be executed.
- Your OLE server is not in the process of starting up, in response to a request from OLE to provide one of your externally creatable objects.

Function RegisterRequest:

```
Public Function RegisterRequest(ClientRequest As Object) As Boolean

'-----  Add object to the queue.  A continuous
'       timer on the main form monitors the
'       queue.
On Error Resume Next
gcolQueue.Add ClientRequest, ClientRequest.ID
If Err Then
  Err = 0
  On Error GoTo 0
  frmClient.List1.AddItem ClientRequest.ID & ": Attempted 2nd request"
  Err.Raise vbObjectError + 512, "CDataServer", "Attempted 2nd Request"

Else
  frmClient.List1.AddItem ClientRequest.ID & ": Request received"
End If

End Function
```

This method is called from the client application using this syntax: *objDataServer.RegisterRequest objRequest*

The reference to the client's object is added to the collection. Each instance of the Client is assigned a unique ID, which is also assigned to the ID property of the client's Request object. This identifies each client in the status messages displayed in the server's listbox. More importantly, it is used to prevent any instance of the client application from ever having more than one request pending in the server.

Error Handling:

The error handler in this example is used to prevent duplicate requests from the same client application. A trappable VB error will occur if attempting to add an object reference to the collection with a duplicate ID. In the sample client application, this occurs if the user clicks on the Request button when a request is already in the Server's queue.

Microsoft recommends that user notification of errors that occur in an OLE server be processed in the client

application. In this example, when a duplicate ID is received, the local error handler is turned off, then the **Raise** method of the Err object is invoked. Because there is no longer an "on error" active in the Server, **Raise** causes an error condition in the client. See **Function Request()** in **frmClient** in CLIENT3.VBP.

Client Project: CLIENT3.VBP

Modules:

- CLIENT.FRM (frmClient)
- REQUEST.CLS (CRequest)

The program creates a form-level object, **objRequest**, of class **CRequest**, which is used to communicate with OLESRV3.EXE.

The program also creates **objDataServer**, a form-level object reference to the **CDataServer** class exposed by the server application.

Basic program operation:

- On program startup, **objRequest** is created.
- In the **Form_Load** event, an instance of **CDataServer** is created.

- If the server is not yet running, as will be the case when the first instance of CLIENT3.EXE is started, creating this object reference will cause server program OLESRV3.EXE to be started. This is because OLESRV3.EXE can be found in the Windows registry; it was registered by VB during the "make exe" process.

- The form contains command button **cmdRequest**. When the button is clicked, **Function Request()** is called (see below). This function calls the **RegisterRequest** method of **objDataServer**, passing a reference to **objRequest** as a parameter. A status message as to the success or failure of the **RegisterRequest** method is displayed and the procedure ends.

- At this point, the request has been "handed off" to the server application. The asynchronous nature of the project is demonstrated here; rather than wait for the Server to return the requested data, processing in the client application continues. A Timer control continuously displays the time in a label on the form, and the user can enter text in the form's text box, while the data request is waiting in the server's queue to be processed.

- When the server application finally gets around to processing the data request, it calls **objRequest**'s **Notify** event, passing the retrieved value as a parameter. The code in **Public Sub Notify**, which merely displays the returned value in a label, executes in the background, while the clock continues to run and the user continues to type in the textbox.

Registering a Request

```
Public Function Request() As Boolean
On Error Resume Next
objDataServer.RegisterRequest objRequest
If Err Then
  Label1 = "Error: " & Err.Description
  On Error GoTo 0
  Request = False
Else
  Request = True
  Label1 = "Request in process"
End If

End Function
```

The statement *objDataServer.RegisterRequest objRequest* calls the RegisterRequest method of the DataServer

object, passing a reference the Client's **objRequest** object as a parameter. The error handler will handle any errors raised in the Server (see below).

Program Shutdown:
When the client's form is closed by the user, the program ends, destroying both local **objRequest** and the reference to **objDataServer** in the Server. When a client destroys the reference to **objDataServer**, it results in the corresponding instance of the **CDataServer** object being destroyed in the Server, triggering the class's **Terminate** event. A counter is decremented; if no more instances of **CDataServer** exist in the Server (meaning that there are no more instances of CLIENT3.EXE running), the database is closed and the program ends.

Error Handling:
One of the rules in this sample project is that a client can have only one request pending at any given time. However, if the **Request** button is clicked while a request is already pending, the Client will attempt to register another request.

Function **RegisterRequest** in the server raises an error if this attempt is made. This error is trapped in client **Function Request ()**(see above). A message is displayed in the status label and the duplicate request is ignored. Note: as the original request is still pending, the status message will be overwritten with the returned value once the request is processed.

Instantiating the DataServer Object:
Variable **objDataServer** is dimensioned in **frmClient**'s Declarations section:

Private objDataServer As ServerTest3.CDataServer

ServerTest3.CDataServer is the fully-qualified class name (or Programmatic ID). *ServerTest3* is the Project Name from the Server's Tools/Options/Project dialog, and *CDataServer* is the classname of the object being created.

The object is not actually created (instantiated) until this 'Set' statement in the Form Load event executes:

Set objDataServer = New ServerTest3.CDataServer

When the first DataServer object is created , OLESRV3.EXE is run and the **Initialize** event in the class module fires, loading and displaying **frmTimer** as well as opening the database. Notice that the Server's form is displayed before the Client's form appears. To put it another way, when the Server's form appears, it indicates visually that OLESRV3.EXE has been started and that the object's **Initialize** event has been fired. (Another indicator, when using a compiled version of the Server instead of running it from within the VB environment, is the appearance of an ***olesrv3*** button on the Win95 taskbar).

Declaring a variable as a specific class in this manner is an example of early binding, which essentially allows VB to create its OLE references in advance. The alternative would be to declare objDataServer As Object, then use the syntax:

Set objDataServer = CreateObject("ServerTest3.CDataServer")

This should be used only in situations when the object type is not known in advance, or in implementations of VBA which cannot declare an object to be of a specific type.

There are two ways to use early binding. The method described above, *Set [variable] = New [class],* will instantiate an object and execute the class's Initialize event when the 'Set' statement executes. The alternative method is to dimension an object variable As New [class]. *The object will not be instantiated until it is referenced for the first time.*

To demonstrate this concept, modify the code in the Client project as follows (also see comments in the code):

- In the Declarations section of frmClient, change:

Private objDataServer As ServerTest3.CDataServer

to:

*Private objDataServer As **New** ServerTest3.CDataServer*

- In Sub Form_Load of frmClient, comment out the line:

  Set objDataServer = New ServerTest3.CDataServer

- Note that the first time objDataServer is referenced is in this line in Function Request():

  objDataServer.RegisterRequest objRequest.

- Run the Client project. Notice that the Server's form no longer appears, even after the Client's form does.

- Click on the **Request** button. This calls **Function Request()**, which executes the above statement. It is at this time that the Server is run, the DataServer object is instantiated, and the form is displayed.

These two methods can be useful for controlling exactly when objects are instantiated, and in the case of OLE Servers, controlling when the Server application is loaded. For example, it may be possible that the Client application would not always need to request data during a session; using the latter 'delayed instantiation' technique, the Server would never run unless the Client actually requested data.