Visual Basic.NET Object-Oriented Programming

Implementing Client/Server Architectures

Part (I of ?)

(Lecture Notes 5A)

Professor: A. Rodriguez
CHAPTER 1 IMPLEMENTING CLIENT/SERVER APPLICATIONS

1.1 Introduction & Review

1.1.1 Overview

1.1.2 Microsoft .NET Key Technologies to Implement the 5-Tier Application Architecture

1.1.3 Review of Dynamic-Link Libraries (DLL) & In-Process Communication

1.1.4 UnAnchored or Distributed Objects - Cross-Platform Communication (Pass-by-Value)

1.1.5 Review of Anchored Objects - Cross-Platform Communication (Pass-by-Reference)

1.1.6 Sample Program #1 Two-Tier or Three-Tier Client/Server; One Solutions; One Computer

1.1.7 Sample Program #2 – Two-Tier Client/Server; Separate Solutions; One Computer
Chapter 1  Implementing Client/Server Applications

1.1 Introduction & Review

1.1.1 Overview

- In this section, we will discuss what is required to implement a two & three-tiers Client/Server applications.
- We will review the required components or parts that are needed and demonstrate using several examples.
- We won’t be actually implementing a true database two-tier system since this requires that we use a Database Management System, so far we are only using an MS Access MDB File, which is not a separate database process, but runs in the same memory space as the client.

1.1.2 Microsoft .NET Key Technologies to Implement the 5-Tier Application Architecture

- In order to implement our architecture, the Business Objects will need to be distributed; they need to possess the technology to allow them to be sent from one machine to another throughout the network.
- The .NET key technologies that will allow us to implement Distributed Business Objects are:

  1. Class Library Project (DLL) – Business Objects need to be packaged as a Class Library or DLL (Dynamic-Link-Library) in order for them to be portable. This is the first step in creating distributed objects. You can package one or several classes into one DLL container.
  2. Anchored & UnAnchored Business Objects – The Business Objects or Classes we create need to be created as either Anchored or UnAnchored Classes.
  3. Remoting – Subsystem of the .NET Framework that allows .NET applications to interact with each other whether is in one computer or computer to computer communication.
  4. Serialization – Process of converting a complex set of data, such as that contained in an object, into a single data or byte stream. The byte stream can be a stream of byte, text, XML stream etc. Deserialization, is the process of unpacking the byte stream to recreate the original complex data.

- We will be covering these technologies in the next several lectures.

1.1.3 Review of Dynamic-Link Libraries (DLL) & In-Process Communication

- A dynamic Link Library is a container that holds procedures, functions, classes etc. that are available to any program that want to use it during run time. The key is that they are dynamically available!
- The code inside the DLL container can be written in any computer language, but it must be compiled into machine language, thus accessible any application regardless of the language used to create the application.
- DLL’s have the following advantages:

  - Reduced Size, Minimal Maintenance & Distributable

- DLL’s have the following Disadvantage:

  - Complex to distribute & manage, DLL’s Cannot execute on their own: DLL’s must be managed and hosted by an executable client process & run in the same memory space as the process or In-Process Communication. They cannot run on their own. DLL’s are slave to the process that is managing them. See diagram below:
1.1.2 UnAnchored or Distributed Objects - Cross-Platform Communication (Pass-by-Value)

- **Unanchored Objects** are known as **Distributed Objects**. These type of objects can be passed from one process to another process or from one machine to another. **By-Value**. By value means that a copy of the original object is placed on the target machine.
- Since a copy of an object is passed to another machine, the other components of that machine can interact with locally or within the process or In-Process communication.
- Note that the **overhead** of this type of communication is during the copy of the object across the network. But once the object is copied, since it now becomes In-process communication there is practically no overhead when making multiple calls to properties and methods.
- The diagram below illustrates this concept:

When to use Unanchored or Distributed Objects

- **Unanchored** or **Distributed** objects need to move from process to process or from machine to machine.
- In the **Two-Tier**: Fat-Client/Thin-Server or Thin-Client/Fat-Server client/server architectures, the **Business Objects Layer** is a good candidate for classification as unanchored since these objects perform the processing and need to be sent through the network in order to get populated and distribute the Business Logic throughout the network.
In the *Three-tier* and *Web-based* client/server architectures, the *Business Objects Layer* will store the unanchored objects:
Implementing Unanchored Objects

To implement Unanchored Objects we need the following .NET technologies:

- **Class Library Project (DLL)** – Business Objects need to be packaged as a Class Library or DLL (Dynamic-Link-Library).
- **Insert the <Serializable()> underscore** before the class declaration:

```
<Serializable()> 
Public Class MyUnAnchoredClass
End Class
```

- **Remoting** – .NET Subsystem that handles communication between objects across a Network. In this case **By-Value**.

From this point the .NET Framework takes care of the rest.

1.1.4 Review of Anchored Objects - Cross-Platform Communication (Pass-by-Reference)

- Anchored objects stay in one location, but other process can access them by **Reference** or **Pass-by-Reference**, which means a pointer to the object.
- This means that when we want to access the objects from computer to computer what we pass is a reference or pointer and not the original object.
- This type of object is stuck or anchored on the machine or process in which it was created. It never moves, but can be accessed by other processes by reference.
- Calling a **property** or **method** is done across the network and processed by the object via the reference, and the result also sent across the network.

**IMPORTANT!** Note that if need to call an anchored object in another machine and we need to make quite a number of property or method calls, this will involve a large overhead. Because of this, getting information from this type of object is done with one call, where you get everything in one shot, to avoid repeated calls to properties or methods.

The following diagram illustrated anchored objects:
When to use Anchored Objects

- Anchored objects are important, because we can guarantee that they will always run on a specific machine only.
- For example in the two types of Two-Tier Client/Fat Server, and the Three-tier client/server architectures, we note that the **Data Access Business Objects Layer** needs to run on a specific server or Application Server. Also, this layer requires a HOST or an EXECUTABLE Process to host the layer. You have the option of creating your own host (EXE) or use a Web Server to host it.
- In the **Web Based Client/Server architecture**, the Web Server will host both the Business Object and Data Access Business Objects, so we don’t need to concern ourselves with creating an executable since the Web process handles this task. This particular server needs to have access to the database server, therefore objects in this layer should be anchored to that specific machine. In addition, in the Three-tier and Th
Implementing Anchored Objects

To implement Anchored Objects we need the following .NET technologies:

- **Class Library Project (DLL)** – Business Objects need to be packaged as a Class Library or DLL (Dynamic-Link-Library).
- **Inherit from .NET Framework class MarshalByRefObject**:

```csharp
Public Class MyAnchoredClass
    Inherits MarshalByRefObject
End Class
```

- **Remoting** – .NET Subsystem that handles communication between objects across a Network. In this case **By-Reference**.

From this point the .NET Framework takes care of the rest.
1.1.5 DLL’s & Client/Server Architectures

- We need to create DLLs to host our classes.
- The question is how many DLLs are required? Well, this depends on the architecture.
- Let’s take a look at the architectures and the required DLL’s.

Two-Tier Fat-Client/Thin-Server

- For this configuration, since both the Business Objects & Data Access Business Objects are in the same layer, we really only need one DLL, as show in the diagrams:

Two-Tier Thin-Client/Fat-Server

- For this configuration, the Business Objects is located in the client, and the Data Access Business Objects in the Server Machine, therefore it would be practical to create two separate DLL’s:
Three-Tier Client/Server

For this configuration, the Business Objects is located in the client, and the Data Access Business Objects in the Server Machine, therefore it would be practical to create two separate DLL’s:

Web-Base Client/Server

For this configuration, since both the Business Objects & Data Access Business Objects are in the same layer or on the web server therefore we only need one DLL:
Example 1.1 – Implementing Web Service to Host Data Access Business Object Layer

Problem statement:
✶ Create a sample program that demonstrates how to create an Executable Process using a Web Service to host the Data Access Business Objects.
✶ This example shows how to implement the host for the Data Access Business Objects in either a Two-Tier Client/Server application a Three-tier Architecture.

We will use DLLs and the Data Access Business Objects will be configures as Anchored Objects.

In addition, the class we will use will contain code that will indicate process information:
✶ Machine or Computer where Process or Class is running on
✶ Process ID

DLL & Class Requirements
✶ The DLL project class should host the following class: clsAnchoredDataAccessBO

    Class Member Data:
    ✶ None

    Class Member Properties & Methods:
    ✶ The function GetServerName(), which returns the name of the computer the object is running in or being hosted in.
    ✶ The function GetProcessID(), returns the ID of the process itself.

Business Object requirements:
✶ Class Library Project (DLL) – Business Objects need to be packaged as a Class Library or DLL (Dynamic-Link-Library).
✶ Inherit from .NET Framework class MarshalByRefObject

    Public Class MyAnchoredClass
       Inherits MarshalByRefObject
    End Class

✶ Remoting – .NET Subsystem that handles communication between objects across a Network. In this case By-Reference.
**HOW IT'S DONE:**

**Part I – Create The Class Library Project to host the Anchored Class:**

- Before we create the Class Library project, we will take the following approach:
  - Since we are creating a stand-alone DLL for distribution, there is no need to create a regular solution, so we will create an Empty Solution, just for the purpose of creating the DLL Class Library Project.
  - Once the DLL or Class Library Project is created the DLL file is available for distribution or use in other programs.

- The steps and code are as follows:

**Step 1: Open the Visual Studio IDE and invoke the Start Page & Select New Project**

**Step 2: In the New Project Dialog select Visual Studio Solutions**

- Select Visual Studio Solutions, Select Blank Solution, Name of Solution and browse to location to store Solution
- In this example, we name the solution “*ApplicationServer*”. Reason I chose this name is simply to point out that we will use this solution to create the Server Application Executable to host the DataAccess Business Objects. In a Two or Three-Tier Client Server architecture, this host will run on the Server portion of a Client/Server topology.

This step will create a blank solution as shown in the *Solution Explore Window*.

**Step a – Project type**

**Step b – Class Library**

**Step c – Project Name**

**Step d – Project Path**
Step 3: Add a New Class Library Project (DLL)

In the Main Menu, select **File|Add Project|New Project...** In the Project Screen, select Visual Basic Project, Class Library, Name of DLL. Note that the location will automatically point to the Empty Solution:

**Step a – Project type**

**Step b – Class Library**

**Step c – Project Name**

**Step d – Project Path**
Step 4: Add the Data Access Business Object Class to the DLL Project

In the Main Menu, select Project|Add Class... to add a new class to the DLL. Name the class as desired. In this example, I will name the class clsAnchoredDataAccessBO to emphasize the point that this class will be used to create Anchored Data Access Business Objects:

The IDE now shows the DLL with the new Class.
Step 6: Add Code to the Class Module – Data Access Business Object Layer

In the Class Module, begin adding the code for the Data Access Business Object.

Step 1: Open the clsAnchoredDataAccessBO Class

Step 2: Make this class an Anchored Object by adding Inheriting from MarshalByRefObject Class:

Inheriting from the MarshalByRefObject Class will give this class everything it needs to behave like an anchored object.

Option Explicit On

Public Class clsAnchoredDataAccessBO
    Inherits MarshalByRefObject    'Can now be an Anchored Business Object

#Region "Remoting Process Statistics"
    'Method returns Name of Machine
    Public Function GetServerName() As String
        Return System.Environment.MachineName
    End Function

    'Method Returns the ID of the process this class is running in
    Public Function GetProcessID() As Integer
    End Function
#End Region

#Region "Regular Methods"
    'Public Function GetCustomer() As clsDistributedCustomer
    '    Dim objCustomer As New clsDistributedCustomer
    '    objCustomer.Name = "NEW Empty Customer"
    '    Return objCustomer
    'End Function
#End Region
End Class
Step 4: Compile and Build the project.

Step 7: DLL is ready for user

DLL IS NOW CREATED AND READY TO BE USED.

WE CAN NOW DISTRIBUTE THIS DLL IS DESIRED. IF YOU LOOK IN THE [BIN] FOLDER OF THE PROJECT YOU WILL SEE THE DLL FILE:

ANY CLIENTS OR PROCESS THAT WISH TO COMMUNICATE WITH THE DLL MUST ADD THE DLL TO THAT SOLUTION OR PROCESS

IN ADDITION, CLIENTS MUST CREATE A REFERENCE TO THE DLL IN ORDER TO BE ABLE TO USE IT.
Part II – Create The Executable Process to Host the DLL:

In order for the DLL to be used in a Server it needs an executable process to run in. The DLL component encapsulating the Data Access Business Object Layer and the Executable program will host it and allow it to interact with a Server.

We have the following options:

Option 1 – Create a Custom host Executable program. This means we create an executable program such as a Windows Application to host the DLL. This method has its advantages but it also requires that we code all the security & other features to make the Data Access Business Objects available.

Option 2 – Create an empty Web Server to host the DLL. In this method we actually use an empty Web Server. Since its an empty Web Project, it has no Web Forms or Web Services etc. Simply its job is to be an executable server to host the DLL. DLLs cannot run on their own. This is a good method since, all Windows 2000 and XP operating Systems come with the Microsoft Internet Information Server (IIS) built in. So why bother to create an executable, when we can quickly put together a web server to host the DLL.

We will chose the IIS empty Web Server option to host our Data Access Business Objects DLL.

Step 1: Add an Empty Web Project

In the Main Menu, select File|Add Project|New Project... In the Project Screen, select Empty Web Project

In the Location text box, name the Web Project, for this example I will name it IISApplicationServer, just to make the point that we are using IIS web server to host the Application Server Anchored Objects. Note that the location will automatically point to the Web Server localhost folder. The Local Host is actually in the path C:\inetpub\wwwroot

After clicking OK, if you a web project folder will be created in the localhost folder or the path C:\inetpub\wwwroot:
Step 2: View of the IDE

In the IDE note the DLL project and the new IISApplicationServer Project which will host the DLL.

![IDE View of Projects]

Step 3: Select WEB Application as the Startup Project

If you have notice that in the Solution Explorer, the DLL Business Objects Project is in bold typeface. This is because to the Solution, this project is the **Startup Project** by default since it was create first.

The Startup Project is the project designated in the solution to execute and control the application.

You need to set the **Web Server Application** as the Startup Project which will control and host the DLL. The Web Server project needs to be the startup since it is the one controlling the DLL. DLL’s cannot execute on its own.

In the Solution explore do the following:

1. In the Solution Explorer Window, Right-Click on the Client Project, in this case the **IISApplicationServer** and select **Set as Startup**
2. You will now notice that the Web Project is highlighted in Bold

![Solution Explorer - IISApplicationServer]
Step 4: IMPORTANT! In the Web Project Add a Reference to the DLL

The first thing we need to do to turn the Empty Web Server to a host, we need to set a reference or pointer to the DLL. This is done as follows:

1. In solution explorer in the Web Server Project, select and EXPAND the References Entry and select Add Reference…
2. Right-Click and select Add References… to invoke the Add Reference screen
3. Select the Projects Tab
4. Select the Business Objects Project:
5. Click the Select button and the DLL will show in the Selected Components section:

6. Click OK and you should see the……..

In the Solution Explore, if you expand the Reference Entry, you will see the reference to the DataAccessBusinessObjectsDLL:
Step 5: Add A Web Configuration File to Web Project

The second step to making the Web Project a host for the DLL is to add a `Web.config` file.

In the Solution Explorer, Right-Click on the Web Project and select `Add New Item...`. In the New Item screen, select `Web Configuration File` and click `Open`.

---

Step 6: View the IDE

A view of the IDE shows the `Web.config` file in the Web Project:
Step 7: Configuring Remoting

- The Web.config file contains XML code.
- We need to add a section configuring the Remoting System to the Web.config file.
- The XML code needs to contain the following information:

- System.runtime.remoting tags, application tags, & service tags:

  ```xml
  <system.runtime.remoting>
  <application>
  <service>
  </service>
  </application>
  </system.runtime.remoting>
  ```

- Within the service tag, add a “wellknown element identifies the following:
  - The mode – how host will handle calls to the object. Set the Mode = “SingleCall”
  - The objectUri – the DLLName.rem. Note that the URI combined with the IIS server name and virtual root form the URL. For our example the URL would look as follows: 
    http://localhost/IISApplicationServer/DataAccessBusinessObjectsDLL.rem
  - The type – The type is a two part statement. The first part is the full name of the class including namespace as follows: Namespace.ClassName. The second part is the name of the DLL.

- The syntax to the complete statement looks as follows:

  ```xml
  <system.runtime.remoting>
  <application>
  <service>
    <wellknown mode="SingleCall"
      objectUri="DLLName.rem"
      type="DLL className, DLL"/>
  </service>
  </application>
  </system.runtime.remoting>
  ```

- Open the Web.Config File and add the remoting configuration code
- Don’t worry at the moment details of this statement. Follow the syntax and add the mode, URI & Type. For our example the XML code looks as follows:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown mode = "SingleCall"
          objectUri="DataAccessBusinessObjectsDLL.rem"
          type="DataAccessBusinessObjectsDLL.clsAnchoredDataAccessBO, DataAccessBusinessObjectsDLL"/>
      </service>
    </application>
  </system.runtime.remoting>
</configuration>
```
Step 8: Testing Application Server with a Browser (IMPORTANT)

- At this point you can test access to the Class within the DLL via the Web Server Process.
- Enter the following URL:
  

- The browser should display a web configuration file and reference to the Business Objects should be seen throughout the file:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="clsAnchoredDataAccessBO">
  <targetNamespace>
  </targetNamespace>
  <wsdlUri>
    http://schemas.xmlsoap.org/wsdl/
  </wsdlUri>
  <instance>
  </instance>
</definitions>
```

- At this point, you have completed the Application Server or host for the Data Access Business Objects using a the IIS Web Server.
- The Application Server is running and waiting for clients to request access.
Part III – Create The Client Process:

✦ Now we create the client process that will access the DLL.
✦ In this example we will create a Form Driven Windows Application to make a request to the Data Access Business Objects running on the IIS Server Process or Application Server.
✦ We will keep this Client application simple. We are interested in understanding the mechanism required to access the Application Server.
✦ In this example, we will add the Client Windows Application in the Same solution. This is may not always be the case and does not has to be so. In the next example we will separate into separate solutions.

Step 1: Add a New Client Project

✦ In the Main Menu, select File|Add Project|New Project… In the Project Screen, select Windows Application. Name the project Note that the location will automatically point to the current solution:

Step 2: View the IDE

✦ The IDE screen now shows the Client Windows application with an empty form as expected. Note the other projects, such as DLL project and IIS Web Server process hosting the DLL:
Step 3: Select Client Application as the Startup Project

- At this point, we want the Client process to be the **Startup Project** for this solution.
- You need to set the **Client Application** as the Startup Project which will control this Solution.
- We don’t have to worry about the Application Server or IIS Web Server. This process already runs on its own. It is a Web Server or Service that is always running.
- In the Solution explorer do the following:

  1. In the Solution Explorer Window, Right-Click on the Client Project, in this case the **WinClient** Application and select **Set as Startup**
  2. You will now notice that the Client project is highlighted in Bold

Step 4: IMPORTANT! In the Client Project, Add a Reference to the DLL

- In order for the Client Project to see the classes in the DLL, it must set a reference or know the Data Type. In other words, the client needs to know about the class in the DLL if its going to call the class methods etc.
- What we are going to do in this example is to simply reference the DLL. This is done as follows:

  1. In solution explorer in the **WinClient** Project section, select and EXPAND the **References** Entry
  2. Right-Click and select **Add References**… to invoke the **Add Reference** screen
  3. Select the **Projects Tab**
  4. Select the **Business Objects DLL Project**:
  5. Click the **Select** button and the DLL will show in the **Selected Components** section:
6. Click OK and note that in the Solution Explorer, there is an entry for the Business Objects reference:

![Solution Explorer](image)

**Step 5: Create Test User Interface**

In the client application, rename the form to `frmTestUI`.

**Step 1:** In the Solution Explore, Right-Click and Rename Form1 to `frmTestUI`

![Form Properties](image)

**Step 2:** Add a Command Button. Name the button `btnProcessInfo`. Set the Text Property to “Get Client/Server Process Info”

**Step 3:** Add a Command Button. Name the button `btnExit`. Set the Text Property to “Exit”. The Form now looks as follows:

![Test Client UI](image)
Step 6: Add Code to the Form’s Event-Handlers

Open the frmTestUI and add code.

Step 1: In the Form Code, code to the btnProcessInfo_Click Event Handler

In this event handler, we will add to code to simply display the process information for the Client process and Server Process Hosting the DLL.

We will use the StringBuilder Class to create a formatted string to display in a message box.

Algorithm is as follows:

1. Create StringBuilder Object
2. Create Server Side Business Object
3. Build the string displaying the process information.
   a) Note that for the Server Process we will call its methods that return process information, such as GetServerName, & GetProcessID.
   b) For the Client Application we will use some of the System Namespace process information classes directly to give us the process information we want.
4. Once the string is built, we will display it via a message box

Code looks as follows:

```vbnet
Private Sub btnProcessInfo_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnProcessInfo.Click
    'Create Object to store a formatted string using the stringbuilder class
    Dim objOutputString As New System.Text.StringBuilder

    'Create Business
    Dim objAnchordObject As New DataAccessBusinessObjectsDLL.clsAnchoredDataAccessBO

    With objOutputString
        'Display Server Process or Process hosting Business Object Information
        .Append("DLL Process Information")
        .Append(vbCrLf) 'Code for Carriage Return or new line
        .Append("Machine Executing DLL: ")
        .Append(objAnchordObject.GetServerName)
        .Append(vbCrLf)
        .Append("DLL Process ID: ")
        .Append(vbCrLf)
        .Append(objAnchordObject.GetProcessID)
        .Append(vbCrLf)

        'Display Client Process Information
        .Append(vbCrLf)
        .Append("Client Process Information")
        .Append(vbCrLf)
        .Append("Machine Executing Client: ")
        .Append(vbCrLf)
        .Append("Client Process ID: ")
        .Append(vbCrLf)
    End With
    MessageBox.Show(objOutputString.ToString)
End Sub
```
Step 2: In the btnExit button Event Handler add code to exit the application

```csharp
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnExit.Click
    Application.Exit()
End Sub
```

Step 7: Set Project’s Properties

In the Solution Explore, Right-Click the Client Project and set the project properties

Step 1: In the Project’s Properties Set the Startup Object to the form frmTestUI
Step 7: View the IDE

The IDE now looks as follows:

Step 8: Build & Execute Project

Step 1: Compile and Build the project.
Step 2: Execute the application.

Step 3: Click the Get Client/Server Process Information button.

A message box with the process information for both client and server should display:

Step 9: Explanation of Results

First let's understand our objectives. We wanted to create a client/server process where the client is running in its own process and the Data Access Business Objects are running in its own separate process.

We have already created the IIS Web Server process to host the DLL. The following diagram represents what the objectives are:
From the diagram, we see that the Client Process is separate and the Server Process as well. We want a two-tier Client/Server application. This architecture will require cross-process communication.

Both these process should have the following attributes:

- **Process ID** – The process ID for these processes should be DIFFERENT since they run in separate processes. We already created a Server IIS Web Server Host Application Server to host the DLL, and this process is separate.
- **Machine Name** – The machine name should be the SAME, since we are running one solution in the same computer.

With this fundamental information let’s analyze what we actually produced.

When we executed the program we got the following results:

![DLL Process Information]

<table>
<thead>
<tr>
<th>DLL Process Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Executing DLL: SATURN</td>
</tr>
<tr>
<td>DLL Process ID: 2036</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Client Process Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Executing Client: SATURN</td>
</tr>
<tr>
<td>Client Process ID: 2036</td>
</tr>
</tbody>
</table>

Summary of results:

- **Machine Name** – The machine name is the SAME. This is OK since we are running from the same machine.
- **Process ID** – The process ID for both server and client process was the SAME: **2036**
  - This means that we were NOT running two separate processes.
  - So this is NOT what we want. Instead of **Two-Tier Client/Server** with two separate process using **cross-process communication** we ended up with a **One-Tier Client/Server** with one process hosting the DLL within that one process or **single-process communication**.
  - What we have is shown in the following One-Tier/One Process application diagram:

![Client Process Memory Space]

**Executable Process - In-Process DLL**
Step 10: Final Comments

- We planned and coded for a two-tier Client Server and we ended up with a One-Tier client Server.
- We actually NEVER were able to connect to the IIS Web Server Application hosting the DLL.
- The DLL we actually connected to was one created by the Client Windows Application.
- The reasons are as follows:
  - Although we created the *IIS Web Server* Application and it is currently running as a Web Server we are NOT connecting to it.
  - This is because we did NOT configure the Client Application to Connect to this Web Server.
  - The object created by the Client is local object NOT the one from the IIS Web Server Host.
  - So, we need to configure the Client to connect to the IIS Web Server Host.
  - We need to configure REMOTING on the Client so that it knows to connect to the IIS Web Server Host.

Step 11: Configure Remoting on the Client Application to Connect to IIS Application Server

- Let configure **Remoting** on the Client.
- In order to invoke remoting in our client we need to do the following:
  1. We need to add a reference to the Remoting sub System from the client process.
  2. Importing the *System.Runtime.Remoting.dll* library or Namespace to our client program.
  3. Call an Object and its methods within the Remoting Library and configure so that the Client knows about the IIS Web Server and the *DataAccessBusinessObjectsDLL.clsAnchoredDataAccessBO*. Note that these methods must be called prior to any processing in the client. This can be done during the loading of the client via the Form_Load() event.

Step 1: Add reference to System.Runtime.Remoting

- The first step is to add a reference to the Remoting subsystem. This is done as follows:
  1. In solution explorer in the Client Project section, select and EXPAND the References
  2. Right-Click and select Add References… to invoke the Add Reference screen
  3. Select the .NET Tab
  4. Scroll and select the *System.Runtime.Remoting*
5. Click the Select button and the Remoting library will show in the Selected Components section:

![Add Reference dialog with Remoting selected](image1.png)

6. Click OK and in the Solution Explorer under references you should see System.Runtime.Remoting reference:

![Solution Explorer with Remoting reference](image2.png)

---

**Step 2: Import the Remoting Namespace in the Form**

The second step is to import the Remoting Namespace in the Client. We will do this on top of the Form. This is done as follows:

```vbnet
Imports System.Runtime.Remoting

Public Class frmTestUI
    Inherits System.Windows.Forms.Form

    ' Code goes here...
```

---
Step 3: Call Remoting Method and Configure to point to IIS Web Server Host and DLL Class

The final step is to call a Remoting Object and its Method from the Remoting Namespace.
In this method you register the the IIS Web Server and the DLL Classes. We will use the RemotingConfiguration object. Then call the RegisterWellKnownClientType method to register the IIS Web Server Process and the DLL Class we wish to use.
This method takes two arguments, the DLL.Class we wish to use and the URL to the IIS Application server. The syntax is as follows:

RemotingConfiguration.RegisterWellKnownClientType(GetType(DLL.Class),"URL")

For our example, the code is as follows:

```vbnet
Private Sub frmTestUI_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    'Call Remoting Object's Method. Arguments are DLL.Class, URL to server
    RemotingConfiguration.RegisterWellKnownClientType( _
        GetType(DataAccessBusinessObjectsDLL.clsAnchoredDataAccessBO), _
        "http://localhost/IISApplicationServer/DataAccessBusinessObjectsDLL.rem")
End Sub
```

Step 12: Build & Execute Project

Step 1: Compile and Build the project.

Step 2: Execute the application.
**Step 3: Click the Get Client/Server Process Information button.**

- A message box with the process information for both client and server should display:

![Message box with process information](image)

**Step 13: Explanation of Results**

- Again, we know our objectives is to create a two-tier Client/Server application and access the Data Access Business Objects in the IIS Web Server Process.

  - From the Message box results we have the following:

    - **Machine Name** – The machine name is the SAME for both the Server and the Client. This is OK since we are running from the same machine.
    - **Server Process ID** – The process ID for server = 3912
    - **Client Process ID** – The process ID for client = 4796

      - This means that we were **ARE** running two separate processes as expected. Both process ID’s are different.
      - This is exactly what we want a **Two-Tier Client/Server** with two separate process using **cross-process communication**.

- We have accomplished our goals, we created the IIS Web Server process to host the DLL and a client process to connect to the host. The following diagram represents what we have accomplished:
Step 14: Remoting Communication – Channel & Formatter (Optional but Recommended)

At this point, we have our client successfully using Remoting to Communicate with the Data Access Business Object in the IIS Web Server Host.

Nevertheless, we now address how the communication is being handled across the processes or through the network.

Earlier we discussed the two types of Channels & Formatters available for communication in remoting:

- **HTTP** – This type of channel sends data formatted in XML or plain TEXT using the SOAP Formatter.
- **TCP** – This type of channel sends data formatted in BINARY.

We cannot use the TCP with Web Servers, only HTTP.

Since we are using the **IIS Web Server**, by default we are using the HTTP channel transmitting in plain TEXT XML.

Transmitting in plain TEXT has the following disadvantages:

- Not very efficient
- Not secure. Transmitting plain text is not good security.

It turns out that IIS Servers support both SOAP and BINARY Formats on the HTTP Channel. It is up to the client to decide which channel to use.

So, with additional configurations we can have our client transmit in BINARY via the HTTP channel.

In order to select the Channel and Formatter, we need to do the following:


2. Create Hashtable, Formatter & Channel objects. Call a Remoting Subsystem method to set the BINARY format and pass these objects as arguments. We will add these objects and method calls to the Form_Load() events

---

**Step 1: Import the additional Remoting Namespaces in the Form**

Import the additional Remoting Namespaces in the Client. We will do this on top of the Form as well:

```csharp
Imports System.Runtime.Remoting
Imports System.Runtime.Remoting.Channels

Public Class frmTestUI
    Inherits System.Windows.Forms.Form

    .........................
```

---

35
Step 2: Add Remoting Objects and configurations

- The second and final step is to add the code.
- The main objectives is to call a Remoting Method named `ChannelServices.RegisterChannel(object)`.
- This method takes one argument, which is the channel object. This object has to be fully configured prior to the call of this method. The method has the following syntax:

  ```
  ChannelServices.RegisterChannel(objchannel)
  ```

- Before we can call this method, we need to create the following:
  1. A HasTable Object with at least a name to assign the channel.
  2. A Formatter Object of type BINARY
  3. A Channel Object using the constructor to pass the name for the channel, and type of formatter object

- For our example, the code is as follows:

```vbnet
Private Sub frmTestUI_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

'Create HasTable Object with the name property of channel
Dim objHashTable As New Hashtable
objHashTable.Item("name") = "HttpBinary"

'Create a Binary formatter object
Dim objformatter As New BinaryClientFormatterSinkProvider

'Create a channel object configured with name & binary formatting
Dim objchannel As New HttpChannel(objHashTable, objformatter, Nothing)

'Call Remoting Method to register channel and Binary format
ChannelServices.RegisterChannel(objchannel)

'RemotingConfiguration.RegisterWellKnownClientType( _
    GetType(DataAccessBusinessObjectsDLL.clsAnchoredDataAccessBO), _
    "http://localhost/IISApplicationServer/DataAccessBusinessObjectsDLL.rem")
```

Step 15: Build & Execute Project

Step 1: Compile and Build the project.
Step 2: Execute the application.

Step 3: Click the Get Client/Server Process Information button.

A message box with the process information for both client and server should display:

- **Machine Name** – The machine name is the SAME for both the Server and the Client. This is OK since we are running from the same machine.
- **Server Process ID** – The process ID for server = 3912
- **Client Process ID** – The process ID for client = 5180

- Two separate process IDs means that we were ARE running two separate processes as expected.

So all we did is format the data being transmitted through the network as BINARY instead of XML TEXT.

From the Message box results we have the following:

- **Server Process ID** – The process ID for server = 3912
- **Client Process ID** – The process ID for client = 5180

We have accomplished our goals, we created the IIS Web Server process to host the DLL and used remoting to enable a client process to connect to the host and we are transmitting in BINARY for better efficient and secure transmission of data.
Our original objectives was to create a Two-Tier Client server process. Actually more of a Thin-Client/Fat-Server Two-Tier Client Server. In addition what is required to also create the Three-Tier-Client Server architecture.

To be more specific, we wanted to spread our distributed application architecture across these client/server architectures as follows:

As show in the Server side of both these architectures, the Data Access Business Objects reside on the Server side. But, this means we need to have a **process to host the Data Access Business objects**. These objects are in a DLL and a DLL cannot run on its own. It needs a process. So we had the following options to host the DLL:

- **Option 1** – Create an Custom host Executable program or a Windows Service.
- **Option 2** – Create an empty Web Server to host the DLL. This was the option we decided

We decided on the second option or **Empty IIS Web Server** because the empty IIS Web Server is available to us, since IIS comes with the Windows 2000/XP OS and has all we need as far as security etc.

So, we ended up creating the following two processes representing the client connecting to the Data Access BO and the IIS Host. Also both process communicate using Binary data transmission:
Our objectives to host the Anchored Data Access Business Objects using an empty IIS Web Server has been accomplished.

But keep in mind. This is **NOT A WEB APPLICATION OR A WEB-BASED CLIENT/SERVER ARCHITECTURE**!

We are simply using an empty Web Server to host our Data Access Business Object so that it can run on a server, so don’t get confused.

To get a full understanding of what has been accomplishes and how it relates to the client/server architectures let’s look at a diagram showing all the processes involved:

In our example, these two-tier processes are running in one computer, but that is usually not the case, this architecture is designed with networking in mind.

Therefore the Client usually runs on one computer, example Window 2000 Professional (Client OS), while the IIS process and the Database Management System (DBMS) process both run on a server computer with a server OS such as Windows 2000 Server. So, two computers are involved, a workstation and a Server machine.
The following diagram illustrates all the processes involved with a Three-Tier Client/Server:

Three-Tier Client Server

- These processes can run on one computer, which is not practical or spread out across the network in different machines.
- Since this architecture is designed with network in mind, these processes are usually in different machines.
- For example, the client can be located on a user's workstation running Windows XP client OS. The Application Server Process can be hosted on a server OS machine running Windows 2000 Server, and finally the Database Management System (DBMS) can be hosted on another server machine running Windows 2000 Server.
- So we are talking about 3 computers involved in the architecture, a Workstation computer, a Server Computer and another Server Computer.
- We will try to demonstrate this in future examples.
In this example we will create a Two-Tier Client/Server Application, but this time we will use two separate solutions. Using two separate solutions is more practical since it reflects the fact that we will run each Tier on a separate machine.

Note that this does not have to be the case, in our last example, although we had one solution, we can still spread out the client executable and run it from a different machine and so on.

But, it may help some students to understand these processes if you use separate solutions.

**Example 4.2 – Two-Tier Client/Server App. Using Separate Solutions**

**Problem statement:**

-create a Two-Tier Client/Server application using the 3-tier Application Architecture as follows:

- **VB.Net Solution Project 1 (Client Application)** – Create a Windows Application Project on its own solution. The UI layer & Business Objects are to be hosted in this solution.
- **VB.Net Solution Project 2 (Client Application)** – Create another solution, and create an Empty Web IIS Project to host the Anchored Data Access Business Objects.
- We will NOT be implementing a Database Layer in this example, but we will make reference to it in the explanation.

Create a DLL or Class Library Project. Place inside the DLL the person class. The class has the following requirements:

**DLL & Class Requirements**

- The DLL project class should host the following class: *clsAnchoredDataAccessBO*
  - *Class Member Data:*
    - None

  *Class Member Properties & Methods:*
  - The function *GetServerName()* , which returns the name of the computer the object is running in or being hosted in.
  - The function *GetProcessID()* , returns the ID of the process itself.

**Business Object requirements:**

- **Class Library Project (DLL)** – Business Objects need to be packaged as a Class Library or DLL (Dynamic-Link-Library).
- Inherit from .NET Framework class MarshalByRefObject:

```csharp
Public Class MyAnchoredClass
  Inherits MarshalByRefObject
End Class
```

- **Remoting** – .NET Subsystem that handles communication between objects across a Network. In this case *By-Reference*. 
HOW IT'S DONE:

Part I – Create The Class Library Project to host the Anchored Class:

- Before we create the Class Library project, we will take the following approach:
  - Since we are creating a stand alone DLL for distribution, there is no need to create a regular solution, so we will create an Empty Solution, just for the purpose of creating the DLL Class Library Project.
  - Once the DLL or Class Library Project is created the DLL file is available for distribution or use in other programs.

- The steps and code are as follows:

Step 1: Open the Visual Studio IDE and invoke the Start Page & Select New Project

Step 2: In the New Project Dialog select Visual Studio Solutions

- Select Visual Studio Solutions, Select Blank Solution, Name of Solution and browse to location to store Solution
- In this example, we name the solution “ApplicationServer”. Reason I chose this name is simply to point out that we will use this solution to create the Server Application Executable to host the DataAccess Business Objects. In a Two or Three-Tier Client Server architecture, this host will run on the Server portion of a Client/Server topology.

This step will create a blank solution as show in the Solution Explore Window:
Step 3: Add a New Class Library Project (DLL)

In the Main Menu, select File|Add Project|New Project… In the Project Screen, select Visual Basic Project, Class Library, Name of DLL. Note that the location will automatically point to the Empty Solution:

- **Step a – Project type**
- **Step b – Class Library**
- **Step c – Project Name**
- **Step d – Project Path**
Step 4: Add the Data Access Business Object Class to the DLL Project

In the Main Menu, select **Project|Add Class...** to add a new class to the DLL. Name the class as desired. In this example, I will name the class *clsAnchoredDataAccessBO* to emphasize the point that this class will be used to create Anchored Data Access Business Objects:

![Add New Item - DataAccessBusinessObjectsDLL](image)

Step 5: IDE is invoked

The IDE now shows the DLL with the new Class.

![IDE Showing DLL with New Class](image)
Step 6: Add Code to the Class Module – Data Access Business Object Layer

In the Class Module, begin adding the code for the Data Access Business Object.

Step 1: Open the clsAnchoredDataAccessBO Class

Step 2: Make this class an Anchored Object by adding Inheriting from MarshalByRefObject Class:

Inheriting from the MarshalByRefObject Class will give this class everything it needs to behave like an anchored object.

Option Explicit On
Public Class clsAnchoredDataAccessBO
    Inherits MarshalByRefObject  'Can now be an Anchored Business Object

#Region "Remoting Process Statistics"
'Method returns Name of Machine
Public Function GetServerName() As String
    Return System.Environment.MachineName
End Function

'Method Returns the ID of the process this class is running in
Public Function GetProcessID() As Integer
End Function
#End Region

#Region "Regular Methods"

'Public Function GetCustomer() As clsDistributedCustomer
'    Dim objCustomer As New clsDistributedCustomer
'    objCustomer.Name = "NEW Empty Customer"
'    Return objCustomer

'End Function
#End Region

End Class
Step 4: Compile and Build the project.

Step 7: DLL is ready for user

- DLL IS NOW CREATED AND READY TO BE USED.

- WE CAN NOW DISTRIBUTE THIS DLL IF DESIRED. IF YOU LOOK IN THE BIN FOLDER OF THE PROJECT YOU WILL SEE THE DLL FILE:

![Image showing the bin folder with a DLL file]

- ANY CLIENTS OR PROCESS THAT WISH TO COMMUNICATE WITH THE DLL MUST ADD THE DLL TO THAT SOLUTION OR PROCESS

- IN ADDITION, CLIENTS MUST CREATE A REFERENCE TO THE DLL IN ORDER TO BE ABLE TO USE IT.
Part II – In the same solution, Add Empty Web Project to Host the Dll:

Now add the IIS empty Web Server option to host our Data Access Business Objects DLL.

Step 1: Open the Visual Studio IDE and invoke the Start Page & Select New Project

Step 2: Add an Empty Web Project

In the Main Menu, select File|Add Project|New Project... In the Project Screen, select Empty Web Project

In the Location text box, name the Web Project, for this example I will name it IISApplicationServer, just to make the point that we are using IIS web server to host the Application Server Anchored Objects. Note that the location will automatically point to the Web Server localhost folder. The Local Host is actually in the path C:\inetpub\wwwroot

After clicking OK, if you a web project folder will be created in the localhost folder or the path C:\inetpub\wwwroot:
Step 3: View of the IDE

In the IDE note the DLL project and the new IISApplicationServer Project which will host the DLL.

Step 4: Select WEB Application as the Startup Project

If you have notice that in the Solution Explorer, the DLL Business Objects Project is in bold typeface. This is because to the Solution, this project is the Startup Project by default since it was create first.

The Startup Project is the project designated in the solution to execute and control the application.

You need to set the Web Server Application as the Startup Project which will control and host the DLL. The Web Server project needs to be the startup since it is the one controlling the DLL. DLL’s cannot execute on its own.

In the Solution explore do the following:

1. In the Solution Explorer Window, Right-Click on the Client Project, in this case the IISApplicationServer and select Set as Startup
2. You will now notice that the Web Project is highlighted in Bold
Step 5: IMPORTANT! In the Web Project Add a Reference to the DLL

The first thing we need to do to turn the Empty Web Server to a host, we need to set a reference or pointer to the DLL. This is done as follows:

7. In solution explorer in the Web Server Project, select and EXPAND the References Entry and select Add Reference…
8. Right-Click and select Add References… to invoke the Add Reference screen
9. Select the Projects Tab
10. Select the Business Objects Project:
11. Click the Select button and the DLL will show in the Selected Components section:

In the Solution Explore, if you expand the Reference Entry, you will see the reference to the DataAccessBusinessObjectsDL:

Click OK and you should see the…….
### Step 6: Add A Web Configuration File to Web Project

The second step to making the Web Project a host for the DLL is to add a *Web.config* file.

- In the Solution Explorer, Right-Click on the Web Project and select *Add New Item...* In the New Item screen, select *Web Configuration File* and click *Open*:

![Add New Item - IISApplicationServer](image)

### Step 7: View the IDE

A view of the IDE shows the *Web.config* file in the Web Project:

![IISApplicationServer - Microsoft Visual Basic.NET (Design) - Web.config](image)
**Step 8: Configuring Remoting**

- The *Web.config* file contains XML code.
- We need to add a section configuring the Remoting System to the *Web.config* file.
- The XML code needs to contain the following information:

  - System.runtime.remoting tags, application tags, & service tags:

    ```xml
    <system.runtime.remoting>
      <application>
        <service>
          <wellknown mode="SingleCall"
                    objectUri="DLLName.rem"
                    type="Namespace.ClassName, DLL"/>
        </service>
      </application>
    </system.runtime.remoting>
    ```

  - Within the service tag, add a “wellknown element identifies the following:
    - The **mode** – how host will handle calls to the object. Set the Mode = “SingleCall”
    - The **objectUri** – the DLLName.rem. Note that the URI combined with the IIS server name and virtual root form the URL. For our example the URL would look as follows:
      `http://localhost/IISApplicationServer/DataAccessBusinessObjectsDLL.rem`
    - The **type** – The type is a two part statement. The first part is the full name of the class including namespace as follows: `Namespace.ClassName`. The second part is the name of the DLL.

  - The syntax to the complete statement looks as follows:

    ```xml
    <system.runtime.remoting>
      <application>
        <service>
          <wellknown mode="SingleCall"
                    objectUri="DataAccessBusinessObjectsDLL.rem"
                    type="DataAccessBusinessObjectsDLL.clsAnchoredDataAccessBO, DataAccessBusinessObjectsDLL"/>
        </service>
      </application>
    </system.runtime.remoting>
    ```

- Open the *Web.Config* File and add the remoting configuration code

- Don’t worry at the moment details of this statement. Follow the syntax and add the mode, URI & Type. For our example the XML code looks as follows:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown mode = "SingleCall"
                    objectUri="DataAccessBusinessObjectsDLL.rem"
                    type="DataAccessBusinessObjectsDLL.clsAnchoredDataAccessBO, DataAccessBusinessObjectsDLL"/>
      </service>
    </application>
  </system.runtime.remoting>
</configuration>
```
**Step 9: Testing Application Server with a Browser (IMPORTANT)**

- At this point you can test access to the Class within the DLL via the Web Server Process.
- Enter the following URL:


- The browser should display a web configuration file and reference to the Business Objects should be seen throughout the file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="clsAnchoredDataAccessBO"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wssd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:soap="http://schemas.xmlsoap.org/soap/"
>
  <!-- Message definitions -->
  <message name="clsAnchoredDataAccessBO.GetServerNameInput" />
  <message name="clsAnchoredDataAccessBO.GetServerNameOutput" />
  <message name="clsAnchoredDataAccessBO.GetProcessIDInput" />
  <message name="clsAnchoredDataAccessBO.GetProcessIDOutput" />

  <!-- PortType definitions -->
  <portType name="clsAnchoredDataAccessBOPortType">
    <operation name="GetServerName">
      <input name="GetServerNameInput"/>
      <output name="GetServerNameResponse"/>
    </operation>
    <operation name="GetProcessID">
      <input name="GetProcessIDInput"/> 
      <output name="GetProcessIDOutput"/>
    </operation>
  </portType>
</definitions>
```

- At this point, you have completed the Application Server or host for the Data Access Business Objects using a IIS Web Server.
- The Application Server is running and waiting for clients to request access.
Client Process Executable

Client Process Memory Space

Client/UI Program

Cross-Process

Web Server Process Memory Space

... DLL encapsulating Anchored

IIS Web Server Service

IIS Web Server Executable Hosting DLL