**MCAD/MCSD Training Guide (70-320): .NET Remoting**

**.NET Remoting Architecture**

.NET remoting enables objects in different application domains to talk to each other. The real strength of remoting is in enabling the communication between objects when their application domains are separated across the network. In this case, remoting transparently handles details related to network communication.

Before I get into details, I'll first answer a basic question: How come remoting is able to establish cross-application domain communication when the application domains do not allow direct calls across their boundaries?

Remoting takes an indirect approach to application domain communication by creating proxy objects as shown in [Figure 3.1](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig01.jpg')). Both application domains communicate with each other by following these steps:

[**[](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig01.jpg'))Figure 3.1**](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig01.jpg')) **In this simplified view of .NET remoting, you can see that client and server communicate indirectly through a proxy object.**

1. When a client object requests an instance of the server object, the remoting system at the client side instead creates a proxy of the server object. The proxy object lives at the client but behaves just like the remote object; this leaves the client with the impression that the server object is in the client's process.
2. When the client object calls a method on the server object, the proxy passes the call information to the remoting system on the client. This remoting system in turn sends the call over the channel to the remoting system on the server.
3. The remoting system on the server receives the call information and, on the basis of it, invokes the method on the actual object on the server (creating the object if necessary).
4. The remoting system on the server collects the result of the method invocation and passes it through the channel to the remoting system on the client.
5. The remoting system at the client receives the server's response and returns the results to the client object through the proxy.

The process of packaging and sending method calls among objects, across application boundaries, via serialization and deserialization, as shown in the preceding steps, is also known as *marshaling*.

Now that you have a basic idea of how .NET remoting works, it's time to get into the details. In the next few sections, I'll explain various key components and terminology of .NET remoting.

**Object Marshaling**

Remotable objects are the objects that can be marshaled across the application domains. In contrast, all other objects are known as non-remotable objects. There are two types of remotable objects:

* Marshal-by-value (MBV) Objects—These objects are copied and passed out of the server application domain to the client application domain.
* Marshal-by-reference (MBR) Objects—A proxy is used to access these objects on the client side. The clients hold just a reference to these objects.

**Create and consume a .NET remoting object**

* **Select a channel protocol and a formatter. Channel protocols include TCP and HTTP. Formatters include SOAP and binary.**

Channels are the objects that transport messages across remoting boundaries such as application domains, processes, and computers. When a client calls a method on the remote objects, the details of the method call—such as parameters and so on—are transported to the remote object through a channel. Any results returned from the remote object are communicated back to the client again through the same channel.

The .NET remoting framework ensures that before a remote object can be called, it has registered at least one channel with the remoting system on the server. Similarly, the client object should specify a channel before it can communicate with a remote object. If the remote object offers more than one channel, the client can connect by using the channel that best suits its requirements.

A channel has two end points. The channel object at the receiving end of a channel (the server) listens to a particular protocol through the specified port number, whereas the channel object at the sending end of the channel (the client) sends information to the receiving end by using the protocol and port number specified by the channel object on the receiving end.

**TIP**

**Port Numbers Should Be Unique on a Machine** Each channel is uniquely associated with a TCP/IP port number. Ports are machine-wide resources; therefore, you cannot register a channel that listens on a port number that is already in use by some other channel on the same machine.

To participate in the .NET remoting framework, the channel object at the receiving end must implement the IChannelReceiver interface, whereas the channel object at the sending end must implement the IChannelSender interface.

The .NET Framework provides implementations for HTTP (Hypertext Transmission Protocol) and TCP (Transmission Control Protocol) channels. If you want to use a different protocol, you can define your own channel by implementing the IChannelReceiver and IChannelSender interfaces.

**HTTP Channels**

The HTTP channels use HTTP for establishing communication between the two ends. These channels are implemented through the classes of the System.Runtime.Remoting.Channels.Http namespace, as shown in Table 3.1.

**Table 3.1: The HTTP Channel Classes**

|  |  |  |
| --- | --- | --- |
| **Class** | **Implements** | **Purpose** |
| HttpServerChannel | IChannelReceiver | An implementation for a server channel that uses the HTTP to receive messages. |
| HttpClientChannel | IChannelSender | An implementation for a client channel that uses the HTTP to send messages. |
| HttpChannel | IChannelReceiver and IChannelSender | An implementation of a combined channel that provides the functionality of both the HttpServerChannel and the HttpClientChannel classes. |

The following code example shows how to register a sender-receiver HTTP channel on port 1234:

using System;

using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Http;

//...

HttpChannel channel = new HttpChannel(1234);

ChannelServices.RegisterChannel(channel);

//...

The ChannelServices class used in the code provides various remoting-related services. One of its static methods is RegisterChannel(), which helps in registering a channel with the remoting framework.

**TCP Channels**

The TCP channel uses TCP for establishing communication between the two ends. The TCP channel is implemented through various classes of the System.Runtime.Remoting.Channels.Tcp namespace, as shown in Table 3.2.

**Table 3.2: The TCP Channel Classes**

|  |  |  |
| --- | --- | --- |
| **Class** | **Implements** | **Purpose** |
| TcpServerChannel | IChannelReceiver | An implementation for a server channel that uses the TCP to receive messages. |
| TcpClientChanel | IChannelSender | An implementation for a client channel that uses the TCP to send messages. |
| TcpChannel | IChannelReceiver and IChannelSender | An implementation of a combined channel that provides the functionality for both TcpServerChannel and TcpClientChannel classes. |

The following code example shows how to register a sender-receiver TCP channel on port 1234:

using System;

using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Tcp;

//...

TcpChannel channel = new TcpChannel(1234);

ChannelServices.RegisterChannel(channel);

//...

**Choosing Between the HTTP and the TCP Channels**

Table 3.3 helps you make a decision about which channel to use in a given scenario. In summary, you'll normally use the TCP channel within a low-risk intranet. For more wide-reach applications, using the HTTP channel makes more sense unless the application's efficiency requirements justify the cost of creating a customized security system.

**TIP**

**Support for Security** .NET remoting has no built-in support for security. It instead depends on the remoting hosts to provide security. The only built-in remoting host that provides security for remote objects is IIS. Therefore, any secured objects must be hosted in IIS.

**Table 3.3: Choosing Between the HTTP Channel and the TCP Channel**

|  |  |  |  |
| --- | --- | --- | --- |
| **Channel** | **Scope** | **Efficiency** | **Security** |
| HttpChannel | **Wide**, using the HTTP channel enables you to host the objects on a robust HTTP server such as IIS. HTTP channels can be used over the Internet, because firewalls do not generally block HTTP communication. | **Less**, because HTTP is a bulky protocol and has lots of extra overhead. | **More**, because when remote objects are hosted in IIS, the HttpChannel can immediately take advantage of Secure Sockets Layer (SSL), Integrated Windows Authentication or Kerberos. |
| TcpChannel | **Narrow**, using the TCP channel over the Internet would require opening certain ports in the firewall and could lead to security breaches. | **More**, because TCP uses raw sockets to transmit data across the network. | **Less**, until you implement a custom security system using the classes provided in the System.Security namespace. |

**Formatters**

**Create and consume a .NET Remoting object**

* **Select a channel protocol and a formatter. Channel protocols include TCP and HTTP. Formatters include SOAP and binary.**

Formatters are the objects that are used to encode and serialize data into messages before they are transmitted over a channel. At the other end of the channel, when the messages are received, formatters decode and deserialize the messages.

To participate in the .NET remoting framework, the formatter classes must implement the IFormatter interface. The .NET Framework packages two formatter classes for common scenarios: the BinaryFormatter class and the SoapFormatter class. If you want to use a different formatter, you can define your own formatter class by implementing the IFormatter interface.

**The SOAP Formatter**

SOAP (Simple Object Access Protocol) is a simple, XML-based protocol for exchanging types and messages between applications. SOAP is an extensible and modular protocol; it is not bound to a particular transport mechanism such as HTTP or TCP.

The SOAP formatter is implemented in the SoapFormatter class of the System.Runtime.Serialization.Formatters.Soap namespace.

SOAP formatting is an ideal way of communicating between applications that use non-compatible architectures. However, SOAP is very verbose. SOAP messages require more bytes to represent the data as compared to the binary format.

**The Binary Formatter**

Unlike SOAP, the binary format used by the .NET Framework is proprietary and can be understood only within .NET applications. However, as compared to SOAP, the binary format of representing messages is very compact and efficient.

The binary formatter is implemented in the BinaryFormatter class of the System.Runtime.Serialization.Formatters.Binary namespace.

**Channels and Formatters**

The HTTP channel uses the SOAP formatter as its default formatter to transport messages to and from the remote objects. The HTTP channel uses SoapClientFormatterSinkProvider and SoapServerFormatterSinkProvider classes to serialize and deserialize messages through the SoapFormatter class. You can create industry-standard XML Web services when using SOAP formatter with the HTTP channel.

The TCP channel uses the binary format by default to transport messages to and from the remote object. The TCP channel uses BinaryClientFormatterSinkProvider and BinaryServerFormatterSinkProvider classes to serialize and deserialize messages through the BinaryFormatter class.

However, channels are configurable. You can configure the HTTP channel to use the binary formatter or a custom formatter rather than the SOAP formatter. Similarly, the TCP channel can be configured to use the SOAP formatter or a custom formatter rather than the binary formatter.

[Figure 3.2](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig02.jpg')) compares the various combinations of channels and formatters on the scale of efficiency and compatibility. You can use this information to decide which combination of channel and formatter you would chose in a given scenario.

[**[Figure 3.2](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig02.jpg'))Figure 3.2**](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig02.jpg')) **A TCP channel with a binary formatter provides maximum efficiency, whereas an HTTP channel with a SOAP formatter provides maximum interoperability.**

**REVIEW BREAK**

* .NET remoting enables objects in different application domains to talk to each other even when they are separated by applications, computers, or the network.
* The process of packaging and sending method calls among the objects across the application boundaries via serialization and deserialization is called marshaling.
* Marshal-by-value (MBV) and Marshal-by-reference (MBR) are the two types of remotable objects. MBV objects are copied to the client application domain from the server application domain, whereas only a reference to the MBR objects is maintained in the client application domain. A proxy object is created at the client side to interact with the MBR objects.
* A channel is an object that transports messages across remoting boundaries such as application domains, processes, and computers. The .NET Framework provides implementations for HTTP and TCP channels to allow communication of messages over the HTTP and TCPs, respectively.
* A channel has two end points. A channel at the receiving end, the server, listens for messages at a specified port number from a specific protocol, and a channel object at the sending end, the client, sends messages through the specified protocol at the specified port number.
* Formatters are the objects that are used to serialize and deserialize data into messages before they are transmitted over a channel. You can format the messages in SOAP or the binary format with the help of SoapFormatter and BinaryFormatter classes in the FCL.
* The default formatter for transporting messages to and from the remote objects for the HTTP channel is the SOAP formatter and for the TCP channel is the binary formatter.

### Remote Object Activation

Only MBR objects can be activated remotely. (*MBV object itself is transferred to the client side)*

**TIP**

Based on the activation mode, MBR objects are classified in the following two categories:

* Server-activated objects
* Client-activated objects

#### Server-Activated Objects

Are those remote objects whose lifetime is directly controlled by the server.

When a client requests an instance of a server-activated object, a proxy to the remote object is created in the client's application domain. The remote object is only instantiated (or activated) on the server when the client calls a method on the proxy object.

The server-activated objects provide limited flexibility because only their default (parameter-less) constructors can be used to instantiate them.

There are two possible activation modes for a server-activated object:

* SingleCall activation mode (ideally for load- balanced environments) No importa el server que procesa las solicitudes
* Singleton activation mode

**NOTE Well-Known Objects** Remote objects activated in SingleCall or Singleton activation mode are also known as server-activated objects or well-known objects.

##### SingleCall

An object is instantiated for the sole purpose of responding to just one client request. After the request is fulfilled, the .NET remoting framework deletes the object and reclaims its memory.

Objects activated in the SingleCall mode are also known as stateless because the objects are created and destroyed with each client request and therefore do not maintain state across the requests. This behavior of the SingleCall mode accounts for greater server scalability as an object consumes server resources for only a small period, therefore allowing the server to allocate resources to other objects.

**TIP**

**Load-Balancing and SingleCall Activation** Sometimes to improve the overall efficiency of an application, the application may be hosted on multiple servers that share the incoming requests to it. In this case, a request can go to any of the available servers for processing. This scenario is called a load-balancing environment.

Because the SingleCall objects are stateless, it does not matter which server processes their requests(). For this reason, SingleCall activation is ideally suited for load- balanced environments.

The SingleCall activation mode is a desired solution when

* The overhead of creating an object is not significant. (Costo de creacion)
* The object is not required to maintain its state.
* The server needs to support a large number of requests for the object.
* The object needs to be supported in a load-balanced environment.

Scenarios that are often well suited for the SingleCall activation mode are those applications where the object is required by the client to do a small amount of work and then the object is no longer required. Some common examples include retrieving the inventory level for an item, displaying tracking information for a shipment, and so on.

##### Singleton

There is at most one instance of the remote object, regardless of the number of clients accessing it.

A Singleton mode object can maintain state information across the method calls. Therefore, they are also sometimes known as stateful objects. The state maintained by the Singleton mode server-activated object is globally shared by all its clients.

A Singleton object does not exist on the server forever. Its lifetime is determined by the lifetime lease of the object.

A Singleton object is a desired solution when

* The overhead of creating an object is substantial.
* The object is required to maintain its state over a prolonged period.
* Several clients need to work on the shared state.

Singleton activation mode is useful in scenarios such as in a chat server where multiple clients talk to the same remote object and share data between one another.

#### Client-Activated Objects

A those remote objects whose lifetime is directly controlled by the client. This is in direct contrast with SAOs, where the server, not the client, has complete control over objects' lifetimes.

Client-activated objects are instantiated on the server as soon as the client requests the object to be created. Unlike SAOs, CAOs do not delay object creation until the first method is called on the object.

You can use any of the available constructors of the remotable class to create a CAO. A typical CAO activation involves the following steps:

1. When the client attempts to create an instance of the server object, an activation request message is sent to the remote server.
2. The server then creates an instance of the requested class by using the specified constructor and returns an ObjRef object to the client application that invoked it. The ObjRef object contains all the required information to generate a proxy object that is capable of communicating with a remote object.
3. The client uses the ObjRef object to create a proxy of the server object on the client side.

An instance of the CAO serves only the client that was responsible for its creation, and the CAO doesn't get discarded with each request. For this reason, a CAO can maintain state with each client that it is serving, but unlike Singleton SAOs, different CAOs cannot share a common state.

The lifetime of a CAO is determined by the lifetime leases. I'll talk more about this topic shortly in a section titled "Lifetime Leases."

A CAO is a desired solution when

* Clients want to maintain a private session with the remote object.
* Clients want to have more control over how the objects are created and how long they will live.

CAOs are useful in scenarios such as entering a complex purchase order where multiple roundtrips are involved and clients want to maintain their own private state with the remote object.

**Comparing the Object Activation Techniques**

Based on the discussions in the previous section, the various object activation techniques can be compared as shown in [Figure 3.3](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig03.jpg')).

[**[Figure 3.3](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig03.jpg'))Figure 3.3**](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig03.jpg')) **The SingleCall server activation offers maximum scalability, whereas the client activation offers maximum flexibility.**

SingleCall server activation mode offers maximum scalability because the remote object occupies server resources for the minimum length of the time. This enables the server to allocate its resources between many clients.

On the other hand, the client activation of remote objects offers maximum flexibility because you have complete control over the construction and lifetime of the remote object.

**Lifetime Leases**

A lifetime lease is the period of time that a particular object shall be active in memory before the .NET framework deletes it and reclaims its memory. Both Singleton SAOs and CAOs use lifetime leases to determine how long they should continue to exist.

**NOTE**

**Leases and Activation Mode** Leases apply only to Singleton SAOs and CAOs. With SingleCall SAOs, objects are created and destroyed with each method call.

**TIP**

**Custom and Infinite Lifetimes** A remote object can choose to have a custom defined lifetime if the InitializeLifetimeService() method of the base class, MarshalByRefObject, is overridden. If the InitializeLifetimeService() method returns a null, the type tells the .NET remoting system that its instances are intended to have an infinite lifetime.

A lifetime lease is represented by an object that implements the ILease interface that is defined in the System.Runtime.Remoting.Lifetime namespace. Some of the important members of this interface are listed in Table 3.4.

**Table 3.4: Important Members of the ILease Interface**

|  |  |  |
| --- | --- | --- |
| **Member Name** | **Type** | **Description** |
| CurrentLeaseTime | Property | Gets the amount of time remaining on the lease before the object is marked for garbage collection. |
| InitialLeaseTime | Property | Gets or sets the initial time for the lease. If the object does not receive any method calls, it lives for only this period. |
| Register() | Method | Registers a sponsor for the lease. |
| Renew() | Method | Renews a lease for the specified time. |
| RenewOnCallTime | Property | Gets or sets the amount of time by which a call to the remote object will increase the CurrentLeaseTime. |
| SponsorshipTimeout | Property | Gets or sets the amount of time to wait for a sponsor to return with a lease renewal time. |

Simply speaking, the lease works as follows:

* When an object is created, the value of the InitialLeaseTime property (which is by default 5 minutes) is used to set its lifetime lease (CurrentLeaseTime).
* Whenever the object receives a call, its CurrentLeaseTime is increased by the time specified by value of RenewOnCallTime property (which is by default 2 minutes).
* The client can also renew a lease for a remote object by directly calling the ILease.Renew() method:
* ILease lease = (ILease)
* RemotingServices.GetLifetimeService(
* RemoteObject);
* TimeSpan expireTime =

lease.Renew(TimeSpan.FromSeconds(60));

* When the value of CurrentLeaseTime reaches 0, the .NET Framework contacts any sponsors registered with the lease to check whether they are ready to sponsor a renewal of the lease of the object.
* If the sponsor does not renew the object or the server cannot contact the sponsor within the duration specified by the SponsorshipTimeout property, then the object is marked for garbage collection.

Sponsors are the objects responsible for dynamically renewing an object's lease if its lease expires. Sponsors implement the ISponsor interface and are registered with the lease manager by calling the ILease.Register() method. When the lease for such an object expires, the lease manager calls the ISponsor.Renewal() method implemented by the sponsor objects to renew the lease time. For more information about sponsors, refer to the "Renewing Leases" topic in the .NET Framework Developer's Guide.

**REVIEW BREAK**

* The MBR remotable objects can be activated in two modes: server-activated mode and client-activated mode.
* Server-activated objects (SAOs) are those remote objects whose lifetime is directly controlled by the server.
* You can activate SAOs in two ways: SingleCall (object is created for each client request) and Singleton (object is created once on the server and is shared by all clients).
* The SingleCall activation mode provides the maximum scalability because it does not maintain any state and the object lives for the shortest duration possible.
* CAOs are created for each client when the client requests that a remote object be created. These objects maintain state for each client with which they are associated.
* The leased-based lifetime process determines how long Singleton SAOs and CAOs should exist.

**Applying .NET Remoting**

So far, I have discussed the architecture and various concepts related to .NET remoting. In this section, you will learn how to apply these concepts to see remoting in action. In particular, you will learn to

* Create a remotable class.
* Create a server-activated object.
* Create a client-activated object.
* Use configuration files to configure the remoting framework.
* Use interface assemblies to compile remoting clients.

**Creating a Remotable Class**

Creating a remotable class is simple. All you need to do is to inherit a class from the MarshalByRefObject class or any of its derived classes. Step-by-Step 3.1 creates a remotable class named DbConnect. This class connects to a specified SQL Server database and enables you to execute a SELECT SQL statement by using its ExecuteQuery() method.

**STEP BY STEP 3.1 - Creating a Remotable Class**

1. Launch Visual Studio .NET. Select File, New, Blank Solution, and name the new solution 320C03. Click OK.
2. Add a new Visual C# .NET class library named StepByStep3\_1 to the solution.
3. In the Solution Explorer, rename the default Class1.cs to DbConnect.cs.
4. Open the DbConnect.cs and replace the code with the following code:
5. using System;
6. using System.Data;
7. using System.Data.SqlClient;
8. namespace StepByStep3\_1
9. {
10. // Marshal-by-Reference Remotable Object
11. public class DbConnect : MarshalByRefObject
12. {
13. private SqlConnection sqlconn;
14. // Default constructor connects to the
15. // Northwind database by calling
16. // the overloaded constructor
17. public DbConnect() : this("Northwind")
18. {
19. }
20. // Parameterized constructor connects to the
21. // specified database
22. public DbConnect(string DbName)
23. {
24. // Open a connection to the specified
25. // sample SQL Server database
26. sqlconn = new SqlConnection(
27. @"data source=(local);" +
28. @"initial catalog=" + DbName +
29. @";integrated security=SSPI");
30. Console.WriteLine(
31. "Created a new connection " +
32. "to the {0} database.", DbName);
33. }
34. public DataSet ExecuteQuery(string strQuery)
35. {
36. Console.Write("Starting to execute " +
37. "the query...");
38. // Create a SqlCommand object
39. // to represent the query
40. SqlCommand sqlcmd =
41. sqlconn.CreateCommand();
42. sqlcmd.CommandType = CommandType.Text;
43. sqlcmd.CommandText = strQuery;
44. // Create a SqlDataAdapter object
45. // to talk to the database
46. SqlDataAdapter sqlda =
47. new SqlDataAdapter();
48. sqlda.SelectCommand = sqlcmd;
49. // Create a DataSet to hold the results
50. DataSet ds = new DataSet();
51. try
52. {
53. // Fill the DataSet
54. sqlda.Fill(ds, "Results");
55. }
56. catch (Exception ex)
57. {
58. Console.WriteLine(ex.Message,
59. "Error executing query");
60. }
61. Console.WriteLine("Done.");
62. return ds;
63. }
64. }

}

1. Select Build, Build StepByStep3\_1. This step packages the remotable class into the file StepByStep3\_1.dll, which is located in the bin\Debug or bin\Release directory of your project. You can navigate to it through the Solution Explorer: Just select the project and click the Show All Files button in the Solution Explorer toolbar.

Although you now have a remotable class available to you, it cannot be called directly from the client application domains yet. For a remotable class to be activated, you need to connect the class to the remoting framework. You'll learn how to do that in the next section.

**Creating a Server-Activated Object**

**Create and consume a .NET Remoting object.**

* **Implement server-activated components.**
* **Instantiate and invoke a .NET Remoting object.**

A remotable class is usually connected with the remoting framework through a separate server program. The server program listens to the client request on a specified channel and instantiates the remote object or invokes calls on it as required.

It is a good idea to keep the remotable class and server program separate; this allows the design to be modular and the code to be reusable.

In this section, I'll show you how to create a remoting server. To achieve this, the remoting server must take the following steps:

1. Create a server channel that listens on a particular port to the incoming object activation requests from other application domains. The following code segment shows how to create a TCP server channel and an HTTP server channel:
2. // Register a TCP server channel on port 1234
3. TcpServerChannel channel = new TcpServerChannel(1234);
4. // Register an HTTP server channel on port 1235

HttpServerChannel channel = new HttpServerChannel(1235);

1. Register the channel with the remoting framework. This registration is performed through the RegisterChannel() method of the ChannelServices class:
2. // Register the channel with the remoting framework

ChannelServices.RegisterChannel(channel);

1. Register the remotable class with the remoting framework. For a server-activated object, the RegisterWellKnownServiceType() method of the RemotingConfiguration class is used to perform this registration, as follows:
2. //Register a remote object with the remoting framework
3. RemotingConfiguration.RegisterWellKnownServiceType(
4. typeof(DbConnect), // type of the remotable class
5. "DbConnect", // URI of the remotable class
6. WellKnownObjectMode.SingleCall //Activation mode

);

1. Here, the first parameter is the type of the remotable class. The second parameter specifies the uniform resource identifier (URI) through which the server publishes the remote object's location. The last parameter specifies the activation mode. The activation mode should be one of the two possible values of the WellKnownObjectMode enumeration—SingleCall and Singleton.

**TIP**

**Accessing an Object Through Multiple Channels** From step 2 and step 3, note that the channel registration and the remote object registration are not related. In fact, a remote object can be accessed through all registered channels.

As I discussed, earlier, an SAO can be activated in two different modes—SingleCall and Singleton. In the next few sections, I'll cover how to create a remoting server for activating objects in each of these modes. I'll also tell the story on the other side of the channel, that is, how to connect the client program to the remoting framework so that it can instantiate the SAO and call methods on it.

**Using the SingleCall Activation Mode to Register a Remotable Class As a Server-Activated Object**

In this section, I'll demonstrate how to create a server that exposes the remotable class through the remoting framework. The server process here is a long-running user interface-less process that continues to listen to incoming client requests on a channel.

Ideally, you should write this type of server program as a Windows service or use an existing Windows service such as Internet Information services (IIS) to work as the remoting server. But I have chosen to write the server program as a console application mainly because I'll use the console window to display various messages that will help you understand the workings of the remoting system.

However, later in this chapter, in the section "Using IIS As an Activation Agent," I cover how to use IIS as a remoting server. I talk about Windows services later in Chapter 6, "Windows Services."

**STEP BY STEP 3.2: Using the SingleCall Activation Mode to Register a Server-Activated Object**

1. Add a new Visual C# .NET console application named StepByStep3\_2 to the solution.
2. In the Solution Explorer, right-click project StepByStep3\_2 and select Add Reference from the context menu. In the Add Reference dialog box, select the .NET tab, select the System.Runtime.Remoting component from the list view, and click the Select button. Now select the Projects tab, select the Project named StepByStep3\_1 (contains the remotable object) from the list view, and click the Select button. Both the selected projects then appear in the Selected Components list, as shown in [Figure 3.4](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig04.jpg')). Click OK.

[**[](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig04.jpg'))Figure 3.4**](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig04.jpg')) **The Add Reference dialog box enables you to add references to components.**

1. In the Solution Explorer, rename the default Class1.cs to DbConnectSingleCallServer.cs. Open the file and change the name of the class to DbConnectSingleCallServer in the class declaration.
2. Add the following using directives:
3. using System.Runtime.Remoting;
4. using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Tcp;

1. Add the following code in the Main() method:
2. [STAThread]
3. static void Main(string[] args)
4. {
5. // Create and register a TCP server channel
6. // that listens on port 1234
7. TcpServerChannel channel =
8. new TcpServerChannel(1234);
9. ChannelServices.RegisterChannel(channel);
10. // Register the service that publishes
11. // DbConnect for remote access in SingleCall mode
12. RemotingConfiguration.RegisterWellKnownServiceType
13. (typeof(StepByStep3\_1.DbConnect), "DbConnect",
14. WellKnownObjectMode.SingleCall);
15. Console.WriteLine("Started server in the " +
16. "SingleCall mode");
17. Console.WriteLine("Press <ENTER> to terminate " +
18. "server...");
19. Console.ReadLine();

}

1. Build the project. This step creates a remoting server that is capable of registering the StepByStep3\_1.DbConnect class for remote invocation by using the SingleCall activation mode.

Step-by-Step 3.2 uses a receiver TCP channel (TcpServerChannel) to register a remotable class with the remoting framework. However, converting this program to use the HTTP channel is not difficult—you just need to change all instances of Tcp to Http.

Step-by-Step 3.2 creates a remoting host that listens on port 1234. This is an arbitrary port number that may or may not work on your computer. A good idea is to check whether a port is already in use by some other application before running this program. You can do this from the command line by using the Windows netstat command.

This suggestion works only in a test scenario; it is not reasonable to instruct a customer to check whether the port is available before starting the application. If the application will run entirely on your company's network, you can safely use a port in the private port range of 49152 through 65535—provided, of course, that the port number you choose is not used by another internal application. If you are distributing the application, you should get a port number registered from IANA (Internet Assigned Numbers Authority). You can see a list of already assigned port numbers at <http://www.iana.org/assignments/port-numbers>.

**Instantiating and Invoking a Server-Activated Object**

At this stage, you have a remotable object as well as a remoting server ready. In this section, I'll show you how to create a remoting client and use it to send messages to the remoting server to activate the remote object. To achieve this, the remoting client needs to take the following steps:

1. Create and register a client channel that is used by the remoting framework to send messages to the remoting server. The type of channel used by the client should be compatible with the channel used by the server. The following examples show how to create a TCP client channel and an HTTP client channel:
2. // Create and register a TCP client channel
3. TcpClientChannel channel = new TcpClientChannel();
4. ChannelServices.RegisterChannel(channel);
5. // Create and register an HTTP client channel
6. HttpClientChannel channel = new HttpClientChannel();

ChannelServices.RegisterChannel(channel);

**TIP**

**Client Channel Registration** You do not specify a port number when you register the client channel. The port number is instead specified at the time of registering the remote class in the client's application domain.

1. Register the remotable class as a valid type in the client's application domain. The RegisterWellKnownClientType() method of the RemotingConfiguration class is used to perform this registration, as shown below:
2. // Register the remote class as a valid
3. // type in the client's application domain
4. RemotingConfiguration.RegisterWellKnownClientType(
5. // Remote class
6. typeof(DbConnect),
7. // URL of the remote class
8. "tcp://*localhost*:1234/DbConnect"

);

1. Here, the first parameter is the type of the remotable class. The second parameter specifies the uniform resource identifier (URI) through which the server publishes the location of the remote object. *Localhost* maps to your local development machine. If the remote object is on some other computer, you'll replace *localhost* with the name of the computer.
2. Instantiate the SAO on the server. You can only use the default constructor.
3. // Instantiate the remote object

DbConnect dbc = new DbConnect();

**TIP**

**Instantiating a Server-Activated Object** You can instantiate an SAO at client side only, by using its default constructor.

I demonstrate the preceding steps in Step-by-Step 3.3, where I create the client program as a Windows application that accepts a SQL statement from the user and passes it to the remotable object. The rows returned by the remotable object are displayed in a DataGrid control.

**STEP BY STEP 3.3: Instantiating and Invoking a Server-Activated Object**

1. Add a new Visual C# .NET Windows application named StepByStep3\_3 to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting and the project StepByStep3\_1 (the remotable class assembly).
3. In the Solution Explorer, rename the default Form1.cs to DbConnectClient.cs. Open the form in code view and change all occurrences of Form1 to refer to DbConnectClient instead.
4. Add the following using directives:
5. using System.Runtime.Remoting;
6. using System.Runtime.Remoting.Channels;
7. using System.Runtime.Remoting.Channels.Tcp;

using StepByStep3\_1;

1. Place two GroupBox controls, a TextBox control (txtQuery, with its MultiLine property set to true), a Button control (btnExecute), and a DataGrid control (dgResults) on the form. Arrange the controls as shown in [Figure 3.5](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig05.jpg')).

[**[](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig05.jpg'))Figure 3.5**](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig05.jpg')) **This form invokes a method of a remote object to execute the given query.**

1. Add the following code in the class definition:
2. // Declare a Remote object

DbConnect dbc;

1. Double-click the form and add the following code in the Load event handler:
2. private void DbConnectClient\_Load(
3. object sender, System.EventArgs e)
4. {
5. // Register a TCP client channel
6. TcpClientChannel channel = new TcpClientChannel();
7. ChannelServices.RegisterChannel(channel);
8. // Register the remote class as a valid
9. // type in the client's application domain
10. RemotingConfiguration.RegisterWellKnownClientType(
11. // Remote class
12. typeof(DbConnect),
13. // URL of the remote class
14. "tcp://localhost:1234/DbConnect"
15. );
16. // Instantiate the remote class
17. dbc = new DbConnect();
18. }

**NOTE**

**Starting Multiple Instances of a Program** When you run a project from the Visual Studio .NET IDE, the default behavior is to run the program in debug mode. However, debug mode does not allow you to start another program from the IDE until you finish the current execution. This can be inconvenient if you want to start multiple client programs from within Visual Studio. A solution is to set the project you want to run first as the startup object and select Debug, Run Without Debugging to run the project. This doesn't lock Visual Studio .NET in the debug mode and you can run more programs from within IDE by using the same technique.

1. Double-click the Button control and add the following code in the Click event handler:
2. private void btnExecute\_Click
3. (object sender, System.EventArgs e)
4. {
5. try
6. { // Invoke a method on the remote object
7. this.dgResults.DataSource =
8. dbc.ExecuteQuery(this.txtQuery.Text);
9. dgResults.DataMember = "Results";
10. }
11. catch(Exception ex)
12. {
13. MessageBox.Show(ex.Message,
14. "Query Execution Error");
15. }

}

1. Right-click on the name of the solution in the Solution Explorer window and select Properties. This opens the Solution '320C03' Property Pages dialog box. In the dialog box, select the Multiple Startup Projects check box and then select the action Start for StepByStep3\_2 and StepByStep3\_3 and set the action to None for other projects, as shown in [Figure 3.6](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig06.jpg')). Make sure that StepByStep3\_2 is placed above StepByStep3\_3, if it is not already in order, and then click on the Move Up and Move Down button to get the correct order.

[**[](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig06.jpg'))Figure 3.6**](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig06.jpg')) **Use the Solution Property Pages dialog box to control what projects are started and in what order.**

1. Build the project. Select Debug, Start to run the project. You should see a command window displaying a message that the server is started in the SingleCall mode.
2. Shortly after, you'll see a Windows form for the client program. Enter a query in the text box and click the Execute Query button. The client invokes a method on the remote object and binds the results from the remote method to the DataGrid control, as shown in [Figure 3.7](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig07.jpg')).

[**[](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig07.jpg'))Figure 3.7**](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig07.jpg')) **The remoting client populates the data grid with the result returned from the method invocation on a remote object.**

In the preceding steps, I have chosen to start the client and server programs within the Visual Studio .NET IDE. You can also start these programs by clicking on StepByStep3\_2.exe and StepByStep3\_3.exe, respectively, from the Windows explorer. Note that the server program should be running before you click on the Execute Query button on the client program.

At the end of Step-by-Step 3.3, when you look at the console window of the server program, note the output as shown in [Figure 3.8](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig08.jpg')). The remote object is created every time you click on the Execute Query button. That's normal behavior based on what you have read so far in the chapter; however, what's peculiar is that for the first call, the constructor is called twice.

[**[Figure 3.8](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig08.jpg'))Figure 3.8**](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig08.jpg')) **The SingleCall remoting server creates a new instance of the remote object with each request.**

In fact, calling the constructor twice for the first request on a server is also a regular behavior of SingleCall object activation. The following two points explain why this happens:

* The first call to the constructor is made by the remoting framework at the server side to check whether it is okay to call this object remotely and to check the object's activation mode.
* The second constructor is called because of the client's call on the remote object. With SingleCall activation, the server does not preserve the state of the constructor that was called earlier and therefore the object has to be re-created with each client request.

Some of you might have a question about why I included a reference to StepByStep3\_1.dll in this project. You may support your argument by saying that the DbConnect class contained in the StepByStep3\_1.dll is a remotable class and its proper place is on the server—not on the client.

Well said, but I have included the reference to StepByStep3\_1.dll for the following reasons:

* The client project StepByStep3\_3 won't compile without it—I am referring to the DbConnect class in the project StepByStep3\_3, and the project StepByStep3\_3 by itself has no definition of DbConnect. When I include a reference to StepByStep3\_1.dll, the project StepByStep3\_3 can resolve the definition for DbConnect from there and enables me to compile the project.
* The client program StepByStep3\_3.exe won't execute without it—I can't remove StepByStep3\_1.dll from the project directory after the compilation is successfully completed. The StepByStep3\_1.dll is required again at the time of running the client. To create the proxy object for the DbConnect class, the CLR must have the metadata that describes DbConnect. This metadata is read from the assembly stored in StepByStep3\_1.dll.

However, you would say this won't work in real life, because the StepByStep3\_1.dll may contain important business logic that you do not want your customers to decompile.

You're right! And I have a solution for that in the form of interface assemblies. With interface assemblies, you just share the interface of an assembly with your customers, not the actual business logic. I'll show you how to create interface assemblies later in this chapter in the section "Using Interface Assemblies to Compile Remoting Clients."

**Using the Singleton Activation Mode to Register a Remotable Class As a Server-Activated Object**

You can activate the same remotable object in different modes without making any changes to the remotable object itself. In the case of SAOs, the choice of activation mode is totally with the server. In this section, I'll show you how to create a remoting server that publishes the DbConnect class as an SAO by using the Singleton activation mode. I'll use the same client program that was created in Step-by-Step 3.3 to test this Singleton server.

**STEP BY STEP 3.4: Using the Singleton Activation Mode to Register a Server-Activated Object**

1. Add a new Visual C# .NET console application named StepByStep3\_4 to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting, and the project StepByStep3\_1 (the remotable class assembly).
3. In the Solution Explorer, rename the default Class1.cs to DbConnectSingletonServer.cs. Open the file and change the name of the class to DbConnectSingletonServer in the class declaration.
4. Add the following using directives:
5. using System.Runtime.Remoting;
6. using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Tcp;

1. Add the following code in the Main() method:
2. [STAThread]
3. static void Main(string[] args)
4. {
5. // Create and Register a TCP server channel
6. // that listens on port 1234
7. TcpServerChannel channel =
8. new TcpServerChannel(1234);
9. ChannelServices.RegisterChannel(channel);
10. // Resgister the service that publishes
11. // DbConnect for remote access in Singleton mode
12. RemotingConfiguration.RegisterWellKnownServiceType
13. (typeof(StepByStep3\_1.DbConnect), "DbConnect",
14. WellKnownObjectMode.Singleton);
15. Console.WriteLine("Started server in the " +
16. "Singleton mode");
17. Console.WriteLine("Press <ENTER> to terminate " +
18. "server...");
19. Console.ReadLine();

}

1. Build the project. This step creates a remoting server that is capable of registering the StepByStep3\_1.DbConnect class for remote invocation by using the Singleton activation mode.
2. Set StepByStep3\_4, the remoting server, as the startup project. Select Debug, Start Without Debugging to run the project. You should see a command window displaying a message that the server is started in the Singleton activation mode.
3. Now set StepByStep3\_3, the remoting client, as the startup project. Select Debug, Start Without Debugging to run the project. Enter a query in the text box and click the button. The code invokes a method on the remote object, which is created when the form loads. The code binds the results from the remote method to the DataGrid control.

Although for the client there has been no changes in the output, if you note the messages generated by the server (see [Figure 3.9](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig09.jpg'))), you can see that just one instance of the connection is created and is shared by all the clients that connect to this server.

[**[Figure 3.9](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig09.jpg'))Figure 3.9**](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig09.jpg')) **The Singleton remoting server uses the same instance of the remote object with subsequent requests.**

**Guided Practice Exercise 3.1**

The objective of this exercise is to create a remoting server that exposes the DbConnect class of StepByStep3\_1 as a Singleton SAO. However, the server and client should communicate via the HTTP channels and SOAP formatter. Other than this, the server and the client programs are similar to those created in Step-by-Step 3.4 and Step-by-Step 3.3, respectively.

How would you use the HTTP channel and SOAP formatter to establish communication between the remoting server and client?

This exercise helps you practice creating a remoting server and client that uses the HTTP channel and SOAP formatter for communication.

You should try working through this problem on your own first. If you get stuck, or if you'd like to see one possible solution, follow these steps:

1. Add a new Visual C# .NET console application named GuidedPracticeExercise3\_1\_Server to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting and the project StepByStep3\_1 (the remotable class assembly).
3. In the Solution Explorer, rename the default Class1.cs to DbConnectSingletonServer.cs. Open the file and change the name of the class to DbConnectSingletonServer in the class declaration.
4. Add the following using directives:
5. using System.Runtime.Remoting;
6. using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Http;

1. Add the following code in the Main() method:
2. [STAThread]
3. static void Main(string[] args)
4. {
5. // Create and Register an HTTP server channel
6. // that listens on port 1234
7. HttpServerChannel channel =
8. new HttpServerChannel(1234);
9. ChannelServices.RegisterChannel(channel);
10. // Register the service that publishes
11. // DbConnect for remote access in Singleton mode
12. RemotingConfiguration.RegisterWellKnownServiceType
13. (typeof(StepByStep3\_1.DbConnect), "DbConnect",
14. WellKnownObjectMode.Singleton);
15. Console.WriteLine("Started server in the " +
16. "Singleton mode");
17. Console.WriteLine("Press <ENTER> to terminate " +
18. "server...");
19. Console.ReadLine();

}

1. Build the project. This step creates a remoting server that is capable of registering the StepByStep3\_1.DbConnect class (the remotable object) for remote invocation by using the Singleton activation mode via the HTTP channel.
2. Add a new Visual C# .NET Windows application named GuidedPracticeExercise3\_1\_Client to the solution.
3. Add references to the .NET assembly System.Runtime.Remoting and the project StepByStep3\_1 (the remotable class assembly).
4. In the Solution Explorer, rename the default Form1.cs to DbConnectClient.cs. Open the form in code view and change all occurrences of Form1 to refer to DbConnectClient instead.
5. Add the following using directives:
6. using System.Runtime.Remoting;
7. using System.Runtime.Remoting.Channels;
8. using System.Runtime.Remoting.Channels.Http;

using StepByStep3\_1;

1. Place two GroupBox controls, a TextBox control (txtQuery, with its MultiLine property set to true), a Button control (btnExecute) and a DataGrid control (dgResults) on the form. Refer to [Figure 3.5](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig05.jpg')) for the design of the form.
2. Add the following code in the class definition:
3. // Declare a Remote object

DbConnect dbc;

1. Double-click the form and add the following code in the Load event handler:
2. private void DbConnectClient\_Load
3. (object sender, System.EventArgs e)
4. {
5. // Register an HTTP client channel
6. HttpClientChannel channel =
7. new HttpClientChannel();
8. ChannelServices.RegisterChannel(channel);
9. // Register the remote class as a valid
10. // type in the client's application domain
11. RemotingConfiguration.RegisterWellKnownClientType(
12. // Remote class
13. typeof(DbConnect),
14. // URL of the remote class
15. "http://localhost:1234/DbConnect"
16. );
17. // Instantiate the remote class
18. dbc = new DbConnect();

}

1. Double-click the Button control and add the following code in the Click event handler:
2. private void btnExecute\_Click
3. (object sender, System.EventArgs e)
4. {
5. try
6. { // Invoke a method on the remote object
7. this.dgResults.DataSource =
8. dbc.ExecuteQuery(this.txtQuery.Text);
9. dgResults.DataMember = "Results";
10. }
11. catch(Exception ex)
12. {
13. MessageBox.Show(ex.Message,
14. "Query Execution Error");
15. }

}

1. Build the project. Set the GuidedPracticeExercise3\_1\_Server, the remoting server, as the startup project. Select Debug, Start Without Debugging to run the project. You should see a command window displaying a message that the server is started in the Singleton activation mode. The remoting server is now ready to receive SOAP messages via HTTP.
2. Now set GuidedPracticeExercise3\_1\_Client, the remoting client, as the startup project. Select Debug, Start Without Debugging to run the project. Enter a query in the text box and click the button. The code invokes a method on the remote object and sends the messages in SOAP format via HTTP.

If you have difficulty following this exercise, review the sections "Channels," "Formatters," "Creating a Remotable Class," and "Creating a Server-Activated Object," earlier in this chapter. Make sure that you also perform Step-by-Step 3.1 to Step-by-Step 3.4. After doing that review, try this exercise again.

**Creating a Client-Activated Object**

**Create and consume a .NET Remoting object.**

* **Implement client-activated components.**
* **Instantiate and invoke a .NET Remoting object.**

When exposing a remotable class as a CAO, no changes are required on the remotable class. Instead, only the server and client differ on how the remotable class is registered with the remoting system.

In this section, I'll show you how to register a remotable class as a CAO and how to instantiate and invoke a CAO from a remoting client.

**Registering a Remotable Class As a Client-Activated Object**

You'll have to take the following steps to register a remotable class as a CAO on the server:

1. Create a server channel that listens on a particular port to the incoming object activation requests from other application domains. The following examples show how to create a TCP server channel and an HTTP server channel:
2. // Register a TCP server channel on port 1234
3. TcpServerChannel channel = new TcpServerChannel(1234);
4. // Register an HTTP server channel on port 1235
5. HttpServerChannel channel =

new HttpServerChannel(1235);

1. Register the channel with the remoting framework. This registration is performed through the RegisterChannel() method of the ChannelServices class:
2. // Register the channel with the remoting framework

ChannelServices.RegisterChannel(channel);

1. Register the remotable class with the remoting framework. For a client-activated object, use the RegisterActivatedServiceType() method of the RemotingConfiguration class to perform the registration, as shown below:
2. // Register a remote object as a CAO
3. // with the remoting framework
4. RemotingConfiguration.RegisterActivatedServiceType(

typeof(DbConnect)); // type of the remotable class

1. Here, the only parameter is the type of the remotable class.

Step-by-Step 3.5 shows how to expose the familiar DbConnect class from StepByStep3\_1 as a CAO.

**STEP BY STEP 3.5: Registering a Remotable Class As a Client-Activated Object**

1. Add a new Visual C# .NET console application named StepByStep3\_5 to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting and the project StepByStep3\_1 (the remotable class assembly).
3. In the Solution Explorer, rename the default Class1.cs to DbConnectCAOServer.cs. Open the file and change the name of the class to DbConnectCAOServer in the class declaration.
4. Add the following using directives:
5. using System.Runtime.Remoting;
6. using System.Runtime.Remoting.Channels;
7. using System.Runtime.Remoting.Channels.Tcp;

using StepByStep3\_1;

1. Add the following code in the Main() method:
2. [STAThread]
3. static void Main(string[] args)
4. {
5. // Create and register a TCP channel on port 1234
6. TcpServerChannel channel =
7. new TcpServerChannel(1234);
8. ChannelServices.RegisterChannel(channel);
9. // Register the client-activated object
10. RemotingConfiguration.RegisterActivatedServiceType
11. (typeof(DbConnect));
12. Console.WriteLine(
13. "Started server in the Client Activation mode");
14. Console.WriteLine(
15. "Press <ENTER> to terminate server...");
16. Console.ReadLine();

}

1. Build the project. This step creates a remoting server that is capable of registering the StepByStep3\_1.DbConnect (the remotable object) class for remote invocation by using the client activation mode.

**Instantiating and Invoking a Client-Activated Object**

To instantiate and invoke a client-activated object, the remoting client needs to take the following steps:

1. Create and register a client channel that the remoting framework uses to send messages to the remoting server. The type of the channel the client uses should be compatible with the channel the server uses. The following examples show how to create a TCP client channel and an HTTP client channel:
2. // Create and register a TCP client channel
3. TcpClientChannel channel = new TcpClientChannel();
4. ChannelServices.RegisterChannel(channel);
5. // Create and register an HTTP client channel
6. HttpClientChannel channel = new HttpClientChannel();

ChannelServices.RegisterChannel(channel);

1. Register the remotable class as a valid type in the client's application domain. This registration is performed using the RegisterActivatedClientType() method of the RemotingConfiguration class as shown below:
2. // Register DbConnect as a type on client,
3. // which can be activated on the server
4. RemotingConfiguration.RegisterActivatedClientType

(typeof(DbConnect), "tcp://localhost:1234");

Here, the first parameter is the type of the remotable class. The second parameter specifies the uniform resource identifier (URI) through which the server publishes the location of the remote object.

**TIP**

**Instantiating a Client-Activated Object** You can instantiate a CAO at the client side by using any of its available constructors.

1. Instantiate the CAO on the server by using the desired constructor.
2. // Instantiate the remote object

DbConnect dbc = new DbConnect("Pubs");

I'll demonstrate the preceding steps in Step-by-Step 3.6. This step is similar to the client program created in Step-by-Step 3.3, but this time the client allows the user to choose between the databases on the server.

**STEP BY STEP 3.6: Instantiating and Invoking a Client-Activated Object**

1. Add a new Visual C# .NET Windows application named StepByStep3\_6 to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting, and the project StepByStep3\_1 (the remotable class assembly).
3. In the Solution Explorer, rename the default Form1.cs to DbConnectClient.cs. Open the form in code view and change all occurrences of Form1 to refer to DbConnectClient instead.
4. Add the following using directives:
5. using System.Runtime.Remoting;
6. using System.Runtime.Remoting.Channels;
7. using System.Runtime.Remoting.Channels.Tcp;

using StepByStep3\_1;

1. Place three GroupBox controls (grpDatabases, grpQuery and grpResults), a ComboBox control (cboDatabases), a TextBox control (txtQuery, with its MultiLine property set to true), two Button controls (btnSelect and btnExecute), and a DataGrid control (dgResults) on the form. Refer to Figure 3.10 for this form's design.
2. Select the Items property of the cboDatabases control in the Properties window and click on the (...) button. This opens the String Collection Editor dialog box. Enter the following database names in the editor:
3. Northwind
4. Pubs

GrocerToGo

1. Click OK to add the databases to the Items collection of the cboDatabases control.
2. Add the following code in the class definition:
3. // Declare a Remote object

DbConnect dbc;

1. Double-click the form and add the following code in the Load event handler:
2. private void DbConnectClient\_Load(
3. object sender, System.EventArgs e)
4. {
5. cboDatabases.SelectedIndex = 0;
6. grpQuery.Enabled = false;

}

1. Double-click the btnSelect control and add the following code in the Click event handler:
2. private void btnSelect\_Click(
3. object sender, System.EventArgs e)
4. {
5. // Disable the Databases group box and
6. // enable the Query group box
7. grpDatabases.Enabled = false;
8. grpQuery.Enabled = true;
9. // Register a TCP client channel
10. TcpClientChannel channel = new TcpClientChannel();
11. ChannelServices.RegisterChannel(channel);
12. // Register the remote class as a valid
13. // type in the client's application domain
14. // by passing the Remote class and its URL
15. RemotingConfiguration.RegisterActivatedClientType
16. (typeof(DbConnect), "tcp://localhost:1234");
17. // Instantiate the remote class
18. dbc = new DbConnect(
19. cboDatabases.SelectedItem.ToString());

}

1. Double-click the btnExecute control and add the following code in the Click event handler:
2. private void btnExecute\_Click
3. (object sender, System.EventArgs e)
4. {
5. try
6. { // Invoke a method on the remote object
7. this.dgResults.DataSource =
8. dbc.ExecuteQuery(this.txtQuery.Text);
9. dgResults.DataMember = "Results";
10. }
11. catch(Exception ex)
12. {
13. MessageBox.Show(ex.Message,
14. "Query Execution Error");
15. }

}

1. Build the project. You now have a remoting client ready to use.
2. Set StepByStep3\_5, the CAO remoting server, as the startup project. Select Debug, Start Without Debugging to run the project. You should see a command window displaying a message that the server is started in the client activation mode.
3. Now set StepByStep3\_6, the remoting client, as the startup project. Select Debug, Start Without Debugging to run the project. Select a database from the combo box and click the Select button. An instance of the remotable object, DbConnect, is created with the selected database. Now enter a query in the text box and click the button. The code invokes a method on the remote object. The code binds the results from the remote method to the DataGrid control as shown in [Figure 3.10](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig10.jpg')).

[**[](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig10.jpg'))Figure 3.10**](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig10.jpg')) **A CAO client allows database selection by taking advantage of the capability to call various constructors.**

1. Again select Debug, Start Without Debugging to run one more instance of the remoting client. Select a different database from the combo box and click the Select button. An instance of the remotable object, DbConnect, is created with the selected database. Now enter a query in the text box and click the button. You should see that the second instance of the client fetches the data from the selected database. Switch to the server command window. You can see that the remote object is instantiated twice with different databases, as shown in [Figure 3.11](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig11.jpg')). This shows that the client activation creates an instance of a remotable object for each client.

[**[Figure 3.11](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig11.jpg'))Figure 3.11**](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig11.jpg')) **The client-activated remoting server creates a new instance of a remote object for each client.**

**REVIEW BREAK**

* To create a remotable class, inherit the remotable class from the MarshalByRefObject class.
* A remotable class is usually connected with the remoting framework through a separate server program. The server program listens to the client request on a specified channel and instantiates the remote object or invokes calls on it as required.
* You should create a server channel that listens on a given port number and register the channel with the remoting framework before you register the remotable class.
* The type of channel the client registers should be compatible with the type of channel the server uses for receiving messages.
* You do not specify a port number when you register the client channel. The port number is instead specified when the remote class is registered in the client's application domain.
* To register SAO objects, on the server side you call the RemotingConfiguration.RegisterWellKnownServiceType() method and on the client side you call the RemotingConfiguration.RegisterWellKnownClientType() method.
* To register CAO objects, on the server side you call the RemotingConfiguration.RegisterActivatedServiceType() method and on the client side you call the RemotingConfiguration.RegisterActivatedClientType() method.
* You can instantiate SAOs only at the client side by using their default constructor, whereas you can instantiate CAOs by using any of the constructors.

**Using Configuration Files to Configure the Remoting Framework**

**Create and consume a .NET Remoting object: Create client configuration files and server configuration files.**

In all the examples written so far, I have written code to register the channel and remote object with the remoting framework. This approach of specifying settings is also known as *programmatic configuration*. Although this approach works fine, there is a drawback: Every time you decide to make any changes in how the channel and the remote objects are registered, you have to recompile the code to see the effect of the changes.

Alternatively, you can store the remoting settings in an XML-based configuration file instead of the code file. The program can automatically pick up any changes made to the configuration file when it executes the next time. You need not recompile the sources. This approach of specifying configuration settings is also known as *declarative configuration*.

The declarative configuration can be specified at two levels:

* Machine-Level—The configuration settings at the machine level can be specified through the machine.config file. The machine.config file is present in the CONFIG subdirectory of the .NET Framework installation directory (typically, Microsoft.NET\Framework\v1.0.3705\CONFIG in the Windows directory of your computer, if you're running version 1.0 of the .NET Framework). Any settings specified in this file apply to all the .NET applications running on the machine.

**NOTE**

**Use Naming Conventions for the Application-Level Configuration Files** Although it is not required, you should prefer using the .NET Framework naming convention for naming the application-level configuration file. This ensures that configuration settings for an application, whether remoting-related or security-related, are all in one place.

* Application-Level—The configuration settings for a specific application can be specified through the application configuration file. In a Windows application, the name of the application–level configuration file includes the full application name and the extension, with .config appended to the extension. For example, the configuration file name for StepByStep3\_7.exe is StepByStep3\_7.exe.config. In an ASP.NET application, the name of the configuration file is web.config.

**TIP**

**Application-Level Configuration File Takes Precedence over Machine-Level Configuration** When you specify both application-level configuration as well as machine-level configuration for an application, the application-level configuration takes priority over the machine-level configuration.

The general format of the configuration file is as follows:

<configuration>

<system.runtime.remoting>

<application>

<lifetime>

<!-- Use this section to specify the -->

<!-- lifetime information for all -->

<!-- the objects in the application. -->

</lifetime>

<service>

<!-- Use this section to specify how a remote-->

<!-- object is exposed by the remoting server-->

<!-- Use the <wellknown> tag to configure-->

<!-- an SAO and use the <activated> tag -->

<!-- to configure a CAO -->

<wellknown />

<activated />

</service>

<client>

<!-- Use this section to specify how a remote-->

<!-- object is consumed by the client -->

<!-- Use the <wellknown> tag to configure a -->

<!-- call to the SAO and use the <activated> -->

<!-- tag to configure a call to the CAO -->

<wellknown />

<activated />

</client>

<channels>

<!-- Use this section to configure the -->

<!-- channels that the application uses -->

<!-- to communicate with the remote objects -->

</channels>

</application>

</system.runtime.remoting>

</configuration>

**TIP**

**Configuration Files** Configuration files are case-sensitive.

**Server-Side Configuration**

In Step-by-Step 3.7, I create a remoting server with Singleton activation mode, similar to one created in Step-by-Step 3.4. However, you'll note that the code itself is reduced, as most of the configuration-related code is now moved to a separate configuration file.

**STEP BY STEP 3.7: Using Configuration Files to Register a Server-Activated Object in the Singleton Activation Mode**

1. Add a new Visual C# .NET console application named StepByStep3\_7 to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting and the project StepByStep3\_1 (the remotable class assembly).
3. In the Solution Explorer, right-click project StepByStep3\_7 and select Add, Add New Item from the context menu. Add an item named StepByStep3\_7.exe.config based on the XML file template.
4. Open the StepByStep3\_7.exe.config file and modify it to contain the following code:
5. <configuration>
6. <system.runtime.remoting>
7. <application>
8. <service>
9. <!-- Set the activation mode,
10. remotable object, and its URL -->
11. <wellknown mode="Singleton"
12. type=
13. "StepByStep3\_1.DbConnect, StepByStep3\_1"
14. objectUri="DbConnect" />
15. </service>
16. <channels>
17. <!-- Set the channel and port -->
18. <channel ref="tcp server"
19. port="1234" />
20. </channels>
21. </application>
22. </system.runtime.remoting>

</configuration>

1. In the Solution Explorer, select the project and click the Show All Files button in the toolbar. Move the StepByStep3\_7.exe.config file from the project folder to the bin\Debug folder under the project where the StepByStep3\_7.exe file will be created when the project is compiled.
2. In the Solution Explorer, rename the default Class1.cs to DbConnectSingletonServer.cs. Open the file and change the name of the class to DbConnectSingletonServer in the class declaration.
3. Add the following using directive:

using System.Runtime.Remoting;

1. Add the following code in the Main() method:
2. [STAThread]
3. static void Main(string[] args)
4. {
5. // Load remoting configuration
6. RemotingConfiguration.Configure
7. ("StepByStep3\_7.exe.config");
8. Console.WriteLine("Started server in the " +
9. "Singleton mode");
10. Console.WriteLine("Press <ENTER> to terminate " +
11. "server...");
12. Console.ReadLine();

}

1. Build the project. This step creates a remoting server that is capable of registering the StepByStep3\_1.DbConnect class for remote invocation by using the Singleton activation mode via its settings in the configuration file.
2. Set StepByStep3\_7, the remoting server, as the startup project. Select Debug, Start Without Debugging to run the project. You should see a command window displaying a message that the server is started in the Singleton activation mode by retrieving the remoting configuration settings from the configuration file.
3. Now set StepByStep3\_3, the remoting client, as the startup project. Select Debug, Start Without Debugging to run the project. Enter a query in the text box and click the button. The code invokes a method on the remote object, which is created when the form loads. The code binds the results from the remote method to the DataGrid control.

The most important thing to note in Step-by-Step 3.7 is the way I have written the <service> and <channels> elements in the configuration file.

The <service> element is written as follows:

<service>

<!-- Set the activcation mode,

remotable object, and its URL -->

<wellknown mode="Singleton"

type=

"StepByStep3\_1.DbConnect, StepByStep3\_1"

objectUri="DbConnect" />

</service>

For an SAO, you need to use the <wellknown> element, where you specify the activation mode, type, and the object URI. The type attribute is specified as a pair made up of the qualified classname (StepByStep3\_1.DbConnect) and the name of the assembly (StepByStep3\_1). The objectUri attribute specifies the endpoint of the URI where the client program will attempt to connect.

The <channels> element can be used to specify the channels the server uses to expose the remotable class. The ref attribute specifies the ID of the channel you want to use. The value ref="tcp server" specifies that the channel is a TCP server channel. If I instead write ref="tcp", the channel becomes a receiver-sender channel.

<channels>

<!-- Set the channel and port -->

<channel ref="tcp server" port="1234" />

</channels>

With the use of configuration files, the remoting code inside the Main() method of the server is now just one statement:

// Load remoting configuration

RemotingConfiguration.Configure(

"StepByStep3\_7.exe.config");

The Configure() method of the RemotingConfiguration class loads the configuration file into memory, parses its contents to locate the <system.runtime.remoting> section, and based on the settings, calls the relevant methods to register the channels and the remoting objects.

**Client-Side Configuration**

The configuration of the remoting client is quite similar to that of the remoting server. However, you configure the <client> element of the configuration file instead of the <service> element.

Step-by-Step 3.8 demonstrates how to use the client-side configuration files.

**STEP BY STEP 3.8: Instantiating and Invoking a Server-Activated Object**

1. Add a new Visual C# .NET Windows application named StepByStep3\_8 to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting, and the project StepByStep3\_1 (the remotable class assembly).
3. In the Solution Explorer, right-click project StepByStep3\_8 and select Add, Add New Item from the context menu. Add an Item named StepByStep3\_8.exe.config, based on the XML file template.
4. Open the StepByStep3\_8.exe.config file and modify it to contain the following code:
5. <configuration>
6. <system.runtime.remoting>
7. <application>
8. <client>
9. <!-- Set the remotable object
10. and its URL -->
11. <wellknown
12. type=
13. "StepByStep3\_1.DbConnect, StepByStep3\_1"
14. url="tcp://localhost:1234/DbConnect"
15. />
16. </client>
17. </application>
18. </system.runtime.remoting>

</configuration>

1. In the Solution Explorer, select the project and click the Show All Files button in the toolbar. Move the StepByStep3\_8.exe.config file from the project folder to the bin\Debug folder under the project, where the StepByStep3\_8.exe file will be created when the project is compiled.
2. In the Solution Explorer, rename the default Form1.cs to DbConnectClient.cs. Open the form in code view and change all occurrences of Form1 to refer to DbConnectClient instead.
3. Place two GroupBox controls (grpQuery and grpResults), a TextBox control (txtQuery, with its MultiLine property set to true), a Button control (btnExecute), and a DataGrid control (dgResults) on the form. Refer to Figure 3.5 for this form's design.
4. Add the following using directives:
5. using System.Runtime.Remoting;

using StepByStep3\_1;

1. Add the following code in the class definition:
2. // Declare a Remote object

DbConnect dbc;

1. Double-click the form and add the following code in the Load event handler:
2. private void DbConnectClient\_Load(
3. object sender, System.EventArgs e)
4. {
5. // Load remoting configuration
6. RemotingConfiguration.Configure
7. ("StepByStep3\_8.exe.config");
8. // Instantiate the remote class
9. dbc = new DbConnect();

}

1. Double-click the Button control and add the following code in the Click event handler:
2. private void btnExecute\_Click(
3. object sender, System.EventArgs e)
4. {
5. try
6. { // Invoke a method on the remote object
7. this.dgResults.DataSource =
8. dbc.ExecuteQuery(this.txtQuery.Text);
9. dgResults.DataMember = "Results";
10. }
11. catch(Exception ex)
12. {
13. MessageBox.Show(ex.Message,
14. "Query Execution Error");
15. }

}

1. Build the project. Set StepByStep3\_7, the remoting server, as the startup project. Select Debug, Start Without Debugging to run the project. You should see a command window displaying a message that the server is started in the Singleton activation mode by retrieving the remoting configuration settings from the configuration file.
2. Set StepByStep3\_8, the remoting client, as the startup project. Select Debug, Start Without Debugging to run the project. Enter a query in the text box and click the button. The code invokes a method on the remote object, which is created from the settings stored in the configuration file. The code binds the results from the remote method to the DataGrid control.

The most important thing to note in Step-by-Step 3.8 is the use of the <client> element in the configuration file. The mode attribute is not part of the <wellknown> element of the client and the objectUri attribute is replaced with the url attribute that specifies the address to connect to the remote object.

<client>

<!-- Set the remotable object

and its URL -->

<wellknown

type=

"StepByStep3\_1.DbConnect, StepByStep3\_1"

url="tcp://localhost:1234/DbConnect"

/>

</client>

Unlike the server configuration file, the <channel> element is not required for the client because the client uses the URL to determine protocol and the port number.

**Guided Practice Exercise 3.2**

In this exercise, you are required to expose the DbConnect class from Step-by-Step 3.1 as a CAO through a remoting server through the HTTP channel. You also need to invoke methods on this client-activated object by creating a form similar to the one created in Step-by-Step 3.6.

However, you should be able to change various parameters such as the channel protocol, port number, URL, name, and type of the remote object without needing to recompile the server.

How would you design such a remoting client and server?

This exercise helps you practice creating configuration files for the remoting server and the client to remote a CAO. You should try working through this problem on your own first. If you get stuck, or if you'd like to see one possible solution, follow these steps:

1. Add a new Visual C# .NET console application named GuidedPracticeExercise3\_2\_Server to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting and the project StepByStep3\_1 (the remotable class assembly).
3. In the Solution Explorer, right-click the project GuidedPracticeExercise3\_2\_Server and select Add, Add New Item from the context menu. Add an Item named GuidedPracticeExercise3\_2\_Server.exe.config based on the XML file template.
4. Open the GuidedPracticeExercise3\_2\_Server.exe.config file and modify it to contain the following code:
5. <configuration>
6. <system.runtime.remoting>
7. <application>
8. <service>
9. <!-- Set the remotable object -->
10. <activated
11. type=
12. "StepByStep3\_1.DbConnect, StepByStep3\_1"
13. />
14. </service>
15. <channels>
16. <channel ref="http" port="1234" />
17. </channels>
18. </application>
19. </system.runtime.remoting>

</configuration>

1. In the Solution Explorer, select the project and click the Show All Files button in the toolbar. Move the GuidedPracticeExercise3\_2\_Server.exe.config file from the project folder to the bin\Debug folder under the project, where the GuidedPracticeExercise3\_2\_Server.exe file will be created when the project is compiled.
2. In the Solution Explorer, rename the default Class1.cs to DbConnectCAOServer.cs. Open the file and change the name of the class to DbConnectCAOServer in the class declaration.
3. Add the following using directive:

using System.Runtime.Remoting;

1. Add the following code in the Main() method:
2. [STAThread]
3. static void Main(string[] args)
4. {
5. // Load remoting configuration
6. RemotingConfiguration.Configure
7. ("GuidedPracticeExercise3\_2\_Server.exe.config");
8. Console.WriteLine(
9. "Started server in the Client Activation mode");
10. Console.WriteLine(
11. "Press <ENTER> to terminate server...");
12. Console.ReadLine();

}

1. Build the project. This step creates a remoting server that is capable of registering the StepByStep3\_1.DbConnect class for remote invocation by using the client activation mode via its settings in the configuration file.
2. Add a new Visual C# .NET Windows application named GuidedPracticeExercise3\_2\_Client to the solution.
3. Add references to the .NET assembly System.Runtime.Remoting and the project StepByStep3\_1 (the remotable class assembly).
4. In the Solution Explorer, right-click the project GuidedPracticeExercise3\_2\_Client and select Add, Add New Item from the context menu. Add an Item named GuidedPracticeExercise3\_2\_Client.exe.config based on the XML file template.
5. Open the GuidedPracticeExercise3\_2\_Client.exe.config file and modify it to contain the following code:
6. <configuration>
7. <system.runtime.remoting>
8. <application>
9. <!-- Set the url for the
10. client activation -->
11. <client url="http://localhost:1234">
12. <!-- Set the remote object
13. type and its assembly -->
14. <activated
15. type=
16. "StepByStep3\_1.DbConnect, StepByStep3\_1"
17. />
18. </client>
19. </application>
20. </system.runtime.remoting>

</configuration>

1. In the Solution Explorer, select the project and click the Show All Files button in the tool bar. Move the GuidedPracticeExercise3\_2\_Client.exe.config file from the project folder to the bin\Debug folder under the project, where the GuidedPracticeExercise3\_2\_Client.exe file will be created when the project is compiled.
2. In the Solution Explorer, rename the default Form1.cs to DbConnectClient.cs. Open the form in code view and change all occurrences of Form1 to refer to DbConnectClient instead.
3. Place three GroupBox controls (grpDatabases, grpQuery and grpResults), a ComboBox control (cboDatabases), a TextBox control (txtQuery, with its MultiLine property set to true), two Button controls (btnSelect and btnExecute), and a DataGrid control (dgResults) on the form. Refer to Figure 3.10 for the design of this form.
4. 17Select the Items property of the cboDatabases control in the Properties window and click on the (...) button. This opens the String Collection Editor dialog box. Enter the following database names in the editor:
5. Northwind
6. Pubs

GrocerToGo

1. Click OK to add the databases to the Items collection of the cboDatabases control.
2. 18Add the following using directives:
3. using System.Runtime.Remoting;

using StepByStep3\_1;

1. Add the following code in the class definition:
2. // Declare a Remote object

DbConnect dbc;

1. Double-click the form and add the following code in the Load event handler:
2. private void DbConnectClient\_Load(
3. object sender, System.EventArgs e)
4. {
5. cboDatabases.SelectedIndex = 0;
6. grpQuery.Enabled = false;

}

1. Double-click the btnSelect control and add the following code in the Click event handler:
2. private void btnSelect\_Click(
3. object sender, System.EventArgs e)
4. {
5. grpDatabases.Enabled = false;
6. grpQuery.Enabled = true;
7. // Load remoting configuration
8. RemotingConfiguration.Configure
9. ("GuidedPracticeExercise3\_2\_Client.exe.config");
10. // Instantiate the remote class
11. dbc = new DbConnect(
12. cboDatabases.SelectedItem.ToString());

}

1. Double-click the btnExecute control and add the following code in the Click event handler:
2. private void btnExecute\_Click(
3. object sender, System.EventArgs e)
4. {
5. try
6. { // Invoke a method on the remote object
7. this.dgResults.DataSource =
8. dbc.ExecuteQuery(this.txtQuery.Text);
9. dgResults.DataMember = "Results";
10. }
11. catch(Exception ex)
12. {
13. MessageBox.Show(ex.Message,
14. "Query Execution Error");
15. }

}

1. Build the project. Set GuidedPracticeExercise3\_2\_Server, the remoting server, as the startup project. Select Debug, Start Without Debugging to run the project. You should see a command window displaying a message that the server is started in the client activation mode by retrieving the remoting configuration settings from the configuration file.
2. Set GuidedPracticeExercise3\_2\_Client, the remoting client, as the startup project. Select Debug, Start Without Debugging to run the project. Enter a query in the text box and click the button. The code invokes a method on the remote object, which is created from the settings stored in the configuration file. The code binds the results from the remote method to the DataGrid control.

If you have difficulty following this exercise, review the sections "Creating a Client-Activated Object" and "Using Configuration Files to Configure the Remoting Framework" earlier in this chapter. After doing that review, try this exercise again.

**Using Interface Assemblies to Compile Remoting Clients**

So far, in all the examples, I had to copy the assembly containing the remotable class to the client project for the client to work. However, this is not a desirable solution in most cases because you may not want to share the implementation of the remotable object with your customers or business partners who are writing the client application.

An advisable solution in that case is to share only the interface of the remotable class with the client application. An interface does not contain any implementation; instead, it just contains the members that are supplied by the class.

In the following sections I demonstrate how you can create interfaces that will enable you to create remote SAOs and CAOs without sharing the implementation. I also introduce the Soapsuds tool, which can help you in automatically creating the interfaces for a remotable class.

**Creating an Interface Assembly**

In this section, I create an assembly that contains an interface named IDbConnect. The interface defines a contract (as shown in Step-by-Step 3.9). All the classes or structs that implement the interface must adhere to this contract.

**STEP BY STEP 3.9: Creating an Interface Assembly**

1. Add a new Visual C# .NET class library named StepByStep3\_9 to the solution.
2. In the Solution Explorer, rename the default Class1.cs to IDbConnect.cs.
3. Open the IDbConnect.cs and replace the code with the following code:
4. using System;
5. using System.Data;
6. namespace StepByStep3\_9
7. {
8. public interface IDbConnect
9. {
10. DataSet ExecuteQuery(string strQuery);
11. }

}

1. Build the project. This step creates an assembly that contains the definition of the IDbConnect interface.

**Creating a Remotable Object That Implements an Interface**

Now that you have created the interface IDbConnect, you can implement this interface in a class named DbConnect. You must ensure that the class DbConnect adheres to the contract defined by the IDbConnect interface. This means you must implement the method ExecuteQuery() and the data types of parameters, and the return value must match with that defined in the interface.

Step-by-Step 3.10 creates the class DbConnect, which adheres to the IDbConnect interface.

**STEP BY STEP 3.10: Creating a Remotable Object That Implements an Interface Assembly**

1. Add a new Visual C# .NET class library named StepByStep3\_10 to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting and the project StepByStep3\_9 (the interface assembly).
3. In the Solution Explorer, rename the default Class1.cs to DbConnect.cs.
4. Open the DbConnect.cs and replace the code with the following code:
5. using System;
6. using System.Data;
7. using System.Data.SqlClient;
8. using StepByStep3\_9;
9. namespace StepByStep3\_10
10. {
11. // Marshal-by-Reference Remotable Object
12. // Inheriting from MarshalByRefObject and
13. // Implementing IDbConnect interface
14. public class DbConnect :
15. MarshalByRefObject, IDbConnect
16. {
17. private SqlConnection sqlconn;
18. // Default constructor connects to the
19. // Northwind database
20. public DbConnect() : this("Northwind")
21. {
22. }
23. // Parameterized constructor connects to the
24. // specified database
25. public DbConnect(string DbName)
26. {
27. // Open a connection to the specified
28. // sample SQL Server database
29. sqlconn = new SqlConnection(
30. "data source=(local);" +
31. "initial catalog=" + DbName +
32. ";integrated security=SSPI");
33. Console.WriteLine(
34. "Created a new connection to " +
35. "the {0} database.", DbName);
36. }
37. public DataSet ExecuteQuery(string strQuery)
38. {
39. Console.Write("Starting to execute " +
40. "the query...");
41. // Create a SqlCommand object
42. // to represent the query
43. SqlCommand sqlcmd =
44. sqlconn.CreateCommand();
45. sqlcmd.CommandType = CommandType.Text;
46. sqlcmd.CommandText = strQuery;
47. // Create a SqlDataAdapter object
48. // to talk to the database
49. SqlDataAdapter sqlda =
50. new SqlDataAdapter();
51. sqlda.SelectCommand = sqlcmd;
52. // Create a DataSet to hold the results
53. DataSet ds = new DataSet();
54. try
55. {
56. // Fill the DataSet
57. sqlda.Fill(ds, "Results");
58. }
59. catch (Exception ex)
60. {
61. Console.WriteLine(ex.Message,
62. "Error executing query");
63. }
64. Console.WriteLine("Done.");
65. return ds;
66. }
67. }

}

1. Build the project. This step creates StepByStep3\_10.DbConnect, a remotable class that implements the IDbConnect interface.

This program is similar to the one created in Step-by-Step 3.1, except that the DbConnect class is now implementing the IDbConnect interface in addition to deriving from the MarshalByRefObject class:

public class DbConnect :

MarshalByRefObject, IDbConnect

{...}

You can expose this remotable object to the clients via the remoting framework by using the techniques that you already know. Step-by-Step 3.11 creates a remoting server that registers the DbConnect class created in Step-by-Step 3.10 as an SAO in Singleton mode.

**STEP BY STEP 3.11: Creating a Remoting Server to Register the Remotable Object That Implements an Interface Assembly**

1. Add a new Visual C# .NET console application named StepByStep3\_11 to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting, the project StepByStep3\_9 (the interface assembly), and StepByStep3\_10 (the remotable class assembly).
3. In the Solution Explorer, rename the default Class1.cs to DbConnectSingletonServer.cs. Open the file and change the name of the class to DbConnectSingletonServer in the class declaration.
4. Add the following using directives:
5. using System.Runtime.Remoting;
6. using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Tcp;

1. Add the following code in the Main() method:
2. [STAThread]
3. static void Main(string[] args)
4. {
5. // Register a TCP server channel that
6. // listens on port 1234
7. TcpServerChannel channel =
8. new TcpServerChannel(1234);
9. ChannelServices.RegisterChannel(channel);
10. // Register the service that publishes
11. // DbConnect for remote access in Singleton mode
12. RemotingConfiguration.RegisterWellKnownServiceType
13. (typeof(StepByStep3\_10.DbConnect), "DbConnect",
14. WellKnownObjectMode.Singleton);
15. Console.WriteLine("Started server in the " +
16. "Singleton mode");
17. Console.WriteLine("Press <ENTER> to terminate " +
18. "server...");
19. Console.ReadLine();

}

1. Build the project. This step creates a remoting server that is capable of registering StepByStep3\_10.DbConnect, the remotable object that implements StepByStep3\_9.IDbConnect interface, for remote invocation by using the Singleton activation mode.

**Creating a Remoting Client That Uses an Interface Instead of the Implementation**

When the remotable class is implementing an interface, you can include just the reference to the interface assembly rather than the implementation assembly itself at the client side. The client then extracts the necessary type information and metadata for compiling and running the program from the interface. Step-by-Step 3.12 shows you how to create such a client.

**STEP BY STEP 3.12: Creating a Remoting Client to Invoke the Remotable Object That Implements an Interface Assembly**

1. Add a new Visual C# .NET Windows application named StepByStep3\_12 to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting and the project StepByStep3\_9 (the interface assembly).
3. In the Solution Explorer, rename the default Form1.cs to DbConnectClient.cs. Open the form in code view and change all occurrences of Form1 to refer to DbConnectClient instead.
4. Add the following using directives:
5. using System.Runtime.Remoting;
6. using System.Runtime.Remoting.Channels;
7. using System.Runtime.Remoting.Channels.Tcp;

using StepByStep3\_9;

1. Place two GroupBox controls (grpQuery and grpResults), a TextBox control (txtQuery, with its MultiLine property set to true), a Button control (btnExecute), and a DataGrid control (dgResults) on the form. Refer to Figure 3.5 for the design of this form.
2. Add the following code in the class definition:
3. // Declare a Remote object

IDbConnect dbc;

1. Double-click the form and add the following code in the Load event handler:
2. private void DbConnectClient\_Load(
3. object sender, System.EventArgs e)
4. {
5. // Register a TCP client channel
6. TcpClientChannel channel = new TcpClientChannel();
7. ChannelServices.RegisterChannel(channel);
8. // Instantiate the remote class
9. dbc = (IDbConnect)
10. Activator.GetObject(typeof(IDbConnect),
11. @"tcp://localhost:1234/DbConnect");

}

1. Double-click the Button control and add the following code in the Click event handler:
2. private void btnExecute\_Click
3. (object sender, System.EventArgs e)
4. {
5. try
6. { // Invoke a method on the remote object
7. this.dgResults.DataSource =
8. dbc.ExecuteQuery(this.txtQuery.Text);
9. dgResults.DataMember = "Results";
10. }
11. catch(Exception ex)
12. {
13. MessageBox.Show(ex.Message,
14. "Query Execution Error");
15. }

}

1. Build the project. Set StepByStep3\_11, the remoting server, as the startup project. Select Debug, Start Without Debugging to run the project. You should see a command window displaying a message that the server is started in the Singleton activation mode.
2. Now, set StepByStep3\_12, the remoting client, as the startup project. Select Debug, Start Without Debugging to run the project. Enter a query in the text box and click the button. The code invokes a method on the remote object, which is created when the form loads. The code binds the results from the remote method to the DataGrid control.

In the preceding program, you add the reference to StepByStep3\_9.dll. This assembly contains the interface IDbConnect. However, an interface variable cannot be instantiated, and therefore you cannot create the instance of the remote object by using the new operator.

But there is an alternative way of creating instances of remote objects, which is especially helpful in the case of interfaces: You can use the methods of the Activator class of the System namespace as follows:

* The Activator.GetObject() Method—Calls the proxy to send messages to the remote object. No messages are sent over the network until a method is called on the proxy. This method is useful for activating the server-activated objects.
* The Activator.CreateInstance() Method—Uses the constructor that best matches the specified parameters to create an instance of the specified type. This method is useful for activating the client-activated objects.

In Step-by-Step 3.12, I am using the Activator.GetObject() method to create a proxy for the server-activated object indicated by the type IDbConnect and the server URL.

**Using the Soapsuds Tool to Automatically Generate an Interface Assembly**

In the previous section, you learned how you can distribute the interface assemblies, rather than the implementation assembly, to the client, and still enable clients to instantiate the remote objects.

However, when you have large numbers of clients, distribution of interface files to each of them is another big issue. However, the .NET Framework SDK provides you the Soapsuds tool (soapsuds.exe) to overcome this issue.

Given the URL for a remotable object, the Soapsuds tool can automatically generate an interface assembly for the remote object. All the clients need to know is the URL of the remotable object. They can then generate interface assemblies on their own by using the Soapsuds tool. A typical usage of the soapsuds.exe is:

soapsuds -nowrappedproxy -urltoschema:http://*MyServer.com*/DbConnect?wsdl -

outputassemblyfile:DbConnectInterface.dll

**NOTE**

**Wrapped Proxies** Wrapped proxies are useful when you need to quickly test a Web service because with the use of wrapped proxies you don't have to write code for channel configuration and remote object registration. However, for better flexibility, you should use unwrapped proxies.

where the urltoschema (or url) switch specifies the remote object's URL, you normally have to append the URL with ?wsdl to enable the Soapsuds tool to generate the metadata from the URL. The outputassemblyfile (or oa) switch specifies the name of the file in which you want the output assembly to be created. The nowrappedproxy (or nwp) switch instructs the Soapsuds tool to generate an unwrapped proxy.

**TIP**

**Soapsuds Tool** The Soapsuds tool can be used only with the HTTP channel. If you are using the TCP channel, you still need to depend on manually creating and distributing the interfaces for remotable classes.

By default the Soapsuds tool generates wrapped proxies. Wrapped proxies are the proxies that store various connection details such as the channel formatting and the URL within the proxy itself. Although this means that you need not write the code for these things, this is not recommended because it reduces the flexibility to change the configuration. If you want to avoid specifying these details in the code, a better way is to use the configuration files.

In Step-by-Step 3.13 you create a client that uses the soapsuds-generated unwrapped proxy to connect with the HTTP server created in Guided Practice Exercise 3.1.

**STEP BY STEP 3.13: Using the Soapsuds Tool (soapsuds.exe) to Automatically Generate an Interface Assembly**

1. Set the GuidedPracticeExercise3\_1\_Server as the startup project. Select Debug, Start Without Debugging to run the project.
2. Select Start, Programs, Microsoft Visual Studio .NET, Visual Studio .NET Tools, Visual Studio .NET Command Prompt to launch a .NET command prompt.
3. Make sure that GuidedPracticeExercise3\_1\_Server is running. Navigate to the StepByStep3\_13 project and run the following command to automatically generate an interface assembly for the remotable object DbConnect, registered by the remoting server, GuidedPracticeExercise3\_1\_Server:
4. soapsuds -url:http://localhost:1234/DbConnect?wsdl

-oa:DbConnectInterface.dll -nowp

1. Add references to the .NET assembly System.Runtime.Remoting and DbConnectInterface.dll (the interface assembly auto-generated by soapsuds.exe in step 3).
2. In the Solution Explorer, rename the default Form1.cs to DbConnectClient.cs. Open the form in code view and change all occurrences of Form1 to refer to DbConnectClient instead.
3. Add the following using directives:
4. using System.Runtime.Remoting;
5. using System.Runtime.Remoting.Channels;
6. using System.Runtime.Remoting.Channels.Http;

using StepByStep3\_1;

1. Place two GroupBox controls (grpQuery and grpResults), a TextBox control (txtQuery, with its MultiLine property set to true), a Button control (btnExecute), and a DataGrid control (dgResults) on the form. Refer to Figure 3.5 for the design of this form.
2. Add the following code in the class definition:
3. // Declare a Remote object

DbConnect dbc;

1. Double-click the form and add the following code in the Load event handler:
2. private void DbConnectClient\_Load(
3. object sender, System.EventArgs e)
4. {
5. // This step is not required if you use
6. // a wrapped proxy
7. // Register an HTTP client channel
8. HttpClientChannel channel =
9. new HttpClientChannel();
10. ChannelServices.RegisterChannel(channel);
11. // This step is not required if you use
12. // a wrapped proxy.
13. // Register the remote class as a valid
14. // type in the client's application domain
15. RemotingConfiguration.RegisterWellKnownClientType(
16. // Remote class
17. typeof(DbConnect),
18. // URL of the remote class
19. "http://localhost:1234/DbConnect");
20. // Instantiate the remote class
21. dbc = new DbConnect();

}

1. Double-click the Button control and add the following code in the Click event handler:
2. private void btnExecute\_Click
3. (object sender, System.EventArgs e)
4. {
5. try
6. { // Invoke a method on the remote object
7. this.dgResults.DataSource =
8. dbc.ExecuteQuery(this.txtQuery.Text);
9. dgResults.DataMember = "Results";
10. }
11. catch(Exception ex)
12. {
13. MessageBox.Show(ex.Message,
14. "Query Execution Error");
15. }

}

1. Build the project. If the GuidedPracticeExercise3\_1\_Server project is not running, set the GuidedPracticeExercise3\_1\_Server as the startup project and select Debug, Start Without Debugging to run the project. You should see a command window displaying a message that the server is started in the Singleton activation mode.
2. Set StepByStep3\_13, the remoting client, as the startup project. Select Debug, Start Without Debugging to run the project. Enter a query in the text box and click the button. The code invokes a method on the remote object, which is created when the form loads. The code binds the results from the remote method to the DataGrid control.

In Step-by-Step 3.13, you learned how to use the Soapsuds tool to automatically generate the interface for the remotable object.

**TIP**

**Soapsuds Tool and Client-Activated Objects** You cannot use the Soapsuds tool with the client-activated object because the metadata generated by the Soapsuds tool can use only the default constructor to create the remote objects.

Also, the interface generated by the Soapsuds tool enables you to directly use the name of the remotable class, which means you can access the remotable class directly in the client as if you had a direct reference to the remotable class at the client side. Therefore you can use the RegisterWellKnownClientType() method of the RemotingConfiguration class, rather than the Activator.GetObject() method, to register the remote object on the client side.

**Creating an Interface Assembly That Works with the Client-Activated Objects**

Both the techniques for generating interface assemblies discussed so far, in the sections, "Creating an Interface Assembly" and "Using the Soapsuds Tool to Automatically Generate an Interface Assembly" are not useful for CAO.

The problem lies with the constructors. The CAOs have capabilities of invoking even the non-default constructors of the remotable class. However, interfaces cannot contain the declaration for a constructor because they contain only method and property declarations.

This common problem is generally solved by following these steps:

1. Create an interface and the remotable class as shown in the previous examples. For easy reference, I'll call them IDbConnect and DbConnect, respectively. IDbConnect contains definitions of all the methods that the DbConnect class wants exposed to the clients.
2. Create an interface that declares as many different methods as there are constructors in the remotable class. Each method is responsible for creating an object in a way defined by its corresponding constructor. This technique is also called as an *abstract factory pattern*. The name contains the word "factory" because it enables you to create objects in different ways. Let's call this interface IDbConnectFactory.
3. Create a class that derives from the MarshalByRefObject class and implements the interface created in step 2 (IDbConnectFactory). Implement all methods to actually create the remotable object (DbConnect) based on the given parameters and return the created instance (DbConnect). Let's call this class DbConnectFactory.
4. Create a remoting server that registers the class created in step 3 (DbConnectFactory) as the remotable class.
5. Create a client that connects to the server and create an instance of the remotable class created in step 3 (DbConnectFactory).
6. Invoke the methods corresponding to a constructor on the remotable object created in step 5. The return value of the constructor is a remotable object of the type defined in step 1 (DbConnect).
7. You now have an object of a type that you originally wanted to remote (DbConnect). You can use this object to invoke methods.

In this section I show you how to implement the preceding technique to create a client-activated object that allows you to use any of its available constructors to create objects.

I use the same remotable class that I defined in Step-by-Step 3.10 (DbConnect), and its interface (IDbConnect), which I defined in Step-by-Step 3.9. I'll start directly with Step 2 of preceding list by creating an IDbConnectFactory interface in Step-by-Step 3.14.

**STEP BY STEP 3.14: Creating an Assembly That Works As an Abstract Factory for the IDbConnect Objects.**

1. Add a new Visual C# .NET class library named StepByStep3\_14 to the solution.
2. Add references to the project StepByStep3\_9 (the interface assembly).
3. In the Solution Explorer, rename the default Class1.cs to IDbConnectFactory.cs.
4. Open the IDbConnectFactory.cs and replace the code with the following code:
5. using System;
6. using System.Data;
7. using StepByStep3\_9;
8. namespace StepByStep3\_14
9. {
10. public interface IDbConnectFactory
11. {
12. IDbConnect CreateDbConnectInstance();
13. IDbConnect CreateDbConnectInstance(
14. string dbName);
15. }

}

1. Build the project. The class library now contains an interface to a class that can act as a factory of objects implementing the IDbConnect interface.

The next step is to create a class that implements the IDbConnectFactory interface and then expose that class as a remotable class through the remoting framework. Step-by-Step 3.15 shows the steps for doing so.

**STEP BY STEP 3.15: Creating a Remoting Server That Exposes the DbConnectFactory As a Remotable Class**

1. Add a new Visual C# .NET console application named StepByStep3\_15 to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting, the project StepByStep3\_9 (the interface assembly containing IDbConnect) StepByStep3\_14 (the new interface assembly containing the IDbConnectFactory) and StepByStep3\_10 (the remotable class assembly).
3. Add a new class to the project that will create another remotable object that inherits from MarshalByRefObject class and implements the IDbConnectFactory interface, created in Step-by-Step 3.14. Name it DbConnectFactory.cs. Open the DbConnectFactory.cs and replace the code with the following code:
4. using System;
5. using StepByStep3\_9;
6. using StepByStep3\_10;
7. using StepByStep3\_14;
8. namespace StepByStep3\_15
9. {
10. class DbConnectFactory :
11. MarshalByRefObject, IDbConnectFactory
12. {
13. public IDbConnect CreateDbConnectInstance()
14. {
15. return new DbConnect();
16. }
17. public IDbConnect CreateDbConnectInstance(
18. string dbName)
19. {
20. return new DbConnect(dbName);
21. }
22. }

}

1. In the Solution Explorer, rename the default Class1.cs to DbConnectFactoryServer.cs. Open the file and change the name of the class to DbConnectFactoryServer in the class declaration.
2. Add the following using directives:
3. using System.Runtime.Remoting;
4. using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Tcp;

1. Add the following code in the Main() method:
2. [STAThread]
3. static void Main(string[] args)
4. {
5. // Register a TCP server channel that
6. // listens on port 1234
7. TcpServerChannel channel =
8. new TcpServerChannel(1234);
9. ChannelServices.RegisterChannel(channel);
10. RemotingConfiguration.RegisterWellKnownServiceType
11. (typeof(DbConnectFactory),
12. "DbConnectFactory",
13. WellKnownObjectMode.Singleton);
14. Console.WriteLine("Started server in " +
15. "client-activated mode");
16. Console.WriteLine("Press <ENTER> to terminate " +
17. "server...");
18. Console.ReadLine();

}

1. Build the project. This step uses the Singleton activation mode to create a remoting server that is capable of registering StepByStep3\_15.DbConnectFactory, the remotable object that implements the StepByStep3\_14. IDbConnectFactory interface, for remote invocation.

Step-by-Step 3.15 exposes the DbConnectFactory as an SAO in Singleton mode. Although the DbConnectFactory object is registered as an SAO, the objects returned by various methods of DbConnectFactory (DbConnect) are CAOs because they are created only on an explicit request from the client.

Step-by-Step 3.16 is the final step for creating a CAO using the abstract factory pattern.

**STEP BY STEP 3.16: Using the Abstract Factory Pattern to Instantiate and Invoke a Client-Activated Object**

1. Add a new Visual C# .NET Windows application named StepByStep3\_16 to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting and the projects StepByStep3\_9 (the IDbConnect interface assembly) and StepByStep3\_14 (the IDbConnectFactory interface assembly).
3. In the Solution Explorer, rename the default Form1.cs to DbConnectClient.cs. Open the form in code view and change all occurrences of Form1 to refer to DbConnectClient instead.
4. Add the following using directives:
5. using System.Runtime.Remoting;
6. using System.Runtime.Remoting.Channels;
7. using System.Runtime.Remoting.Channels.Tcp;
8. using StepByStep3\_9; // contains IDbConnect

using StepByStep3\_14; // contains IDbConnectFactory

1. Place three GroupBox controls (grpDatabases, grpQuery, and grpResults), a ComboBox control (cboDatabases), a TextBox control (txtQuery, with its MultiLine property set to true), two Button controls (btnSelect and btnExecute), and a DataGrid control (dgResults) on the form. Refer to Figure 3.10 for this form's design.
2. Select the Items property of the cboDatabases control in the Properties window and click on the (...) button. This opens the String Collection Editor dialog box. Enter the following database names in the editor:
3. Northwind
4. Pubs

GrocerToGo

Click OK to add the databases to the Items collection of the cboDatabases control.

1. Add the following code in the class definition:
2. // Declare a Remote object

IDbConnect dbc;

1. Double-click the form and add the following code in the Load event handler:
2. private void DbConnectClient\_Load(
3. object sender, System.EventArgs e)
4. {
5. cboDatabases.SelectedIndex = 0;
6. grpQuery.Enabled = false;

}

1. Double-click the btnSelect control and add the following code in the Click event handler:
2. private void btnSelect\_Click(
3. object sender, System.EventArgs e)
4. {
5. // Disable the Databases group box and
6. // enable the Query group box
7. grpDatabases.Enabled = false;
8. grpQuery.Enabled = true;
9. // Register a TCP client channel
10. TcpClientChannel channel = new TcpClientChannel();
11. ChannelServices.RegisterChannel(channel);
12. // Register the remote class as a valid
13. // type in the client's application domain
14. // by passing the Remote class and its URL
15. IDbConnectFactory dbcf = (IDbConnectFactory)
16. Activator.GetObject(typeof(IDbConnectFactory),
17. "tcp://localhost:1234/DbConnectFactory");
18. dbc = dbcf.CreateDbConnectInstance
19. (cboDatabases.SelectedItem.ToString());

}

1. Double-click the btnExecute control and add the following code in the Click event handler:
2. private void btnExecute\_Click
3. (object sender, System.EventArgs e)
4. {
5. try
6. { // Invoke a method on the remote object
7. this.dgResults.DataSource =
8. dbc.ExecuteQuery(this.txtQuery.Text);
9. dgResults.DataMember = "Results";
10. }
11. catch(Exception ex)
12. {
13. MessageBox.Show(ex.Message,
14. "Query Execution Error");
15. }

}

1. Build the project. This step creates a remoting client that is capable of activating DbConnect as a CAO.
2. Set StepByStep3\_15, the remoting server, as the startup project. Select Debug, Start Without Debugging to run the project. You should see a command window displaying a message that the server is started in client-activation mode.
3. Set StepByStep3\_16, the remoting client, as the startup project. Select Debug, Start Without Debugging to run the project. Select a database from the combo box and click the Select button. An instance of the remotable object, DbConnect is created with the selected database. Now enter a query in the text box and click the button. The code invokes a method on the remote object. The code binds the results from the remote method to the DataGrid control.
4. Again select Debug, Start Without Debugging to run the remoting client. Select a different database from the combo box and click the Select button. An instance of the remotable object, DbConnect, is created with the selected database. Now enter a query in the text box and click the button. You should see that the second instance of the client fetches the data from the selected database. Now switch to the Server command window. You see that two times the remote object is instantiated with different databases, as shown earlier in Figure 3.10. This shows that client activation creates an instance of the remotable object per client.

In Step-by-Step 3.16, it is important to note that although the object for DbConnectFactory is created as a server-activated object, the DbConnect object is always created as the client-activated object. For the DbConnectFactory object, there is only one instance for all the clients; however, there is one DbConnect object for each client.

**REVIEW BREAK**

* The .NET Framework allows you to store remoting configuration details in an XML-based configuration file rather than the code file. This causes any changes in the configuration file to be automatically picked up and eliminates the need to recompile the code files.
* To configure remoting configuration details from configuration files, you should call the RemotingConfiguration.Configure() method and pass the configuration file's name.
* You can distribute the interface assemblies to the client rather than the implementation assembly by creating an interface that defines the contract and exposes the member definitions to the client. The remotable class should implement this interface.
* You can use the Soapsuds tool to automatically generate the interface for the remotable object rather than manually define the interface. However, the Soapsuds tool works only when the HTTP channel is used for communication.
* To create an interface that allows client-activated remote objects to be created, you should create an additional interface that declares as many different methods as there are constructors in the remotable class. Each of these methods should be able to create the object in a way that its corresponding constructor defines.

**Using IIS As an Activation Agent**

**Create and consume a .NET Remoting object: Create the listener service.**

So far in this chapter, you are hosting the remotable class by creating your own server that is a console application. The major disadvantage with this approach is that you have to manually start the server if it is not already running.

As discussed before, remoting provides two alternatives to overcome this disadvantage: You can either run the server process as a Windows service instead of a console application, or use IIS as an activation agent (which is a built-in Windows service) for the server process. I'll talk about the former in Chapter 6, and the latter in this section.

Using IIS as an activation agent offers the following advantages:

* You need not write a separate server program to register the remotable classes.
* You need not worry about finding an available port for your server application. You can just host the remotable object and IIS automatically uses port 80.
* IIS can provide other functionality, such as authentication and secure socket layers (SSL).

The following list specifies what you need to do to host a remotable class in IIS:

* Place the assembly containing the remotable objects into the \bin directory of an IIS Web application or place the assembly in the GAC (Global Assembly Cache).

**TIP**

**IIS Does Not Support CAO** When creating IIS-hosted remote objects, you cannot specify constructor parameters; therefore it is not possible to use IIS to activate CAO.

* Configure the remoting settings by placing the <system. runtime.remoting> configuration section into the web.config file for the Web application. Alternatively, you can write the configuration code in the Application\_Start() event handler of the global.asax file in the same way you would register a remote object in an .exe host.
* You should not specify a channel. IIS already listens on port 80. Specifying a port for a channel causes exceptions to be thrown when new IIS worker processes are started.

**NOTE**

**Channels and IIS Activation** IIS Activation supports only HTTP channels. The default formatting is SOAP, but IIS also supports binary and other custom formatting.

* The well-known object URIs must end with .rem or .soap because these are the two extensions that are registered with both IIS (via the aspnet\_isapi.dll) as well as the remoting system (in machine.config).

Step-by-Step 3.17 demonstrates how to use IIS for activating the DbConnect remotable class from Step-by-Step 3.10.

**STEP BY STEP 3.17: Using IIS As an Activation Agent**

1. Add a new empty Web project named StepByStep3\_17 to the solution.
2. Add references to the project StepByStep3\_9 (the interface assembly) and StepByStep3\_10 (the remotable object) by selecting Add Reference from the context menu.
3. Add a Web configuration file to the project. Open the web.config file and add the following <system.runtime.remoting> element inside the <configuration> element:
4. <configuration>
5. <system.runtime.remoting>
6. <application>
7. <service>
8. <!-- Set the activation mode,
9. remotable object, and its URL -->
10. <wellknown mode="Singleton"
11. type="StepByStep3\_10.DbConnect, StepByStep3\_10"
12. objectUri="DbConnect.rem"
13. />
14. </service>
15. </application>
16. </system.runtime.remoting>
17. ...

</configuration>

1. IIS is now hosting the StepByStep3\_10.DbConnect remotable class as a server-activated object by using the Singleton activation mode.

Note that the objectUri of the SAO in the preceding example ends with the extension .rem.

Step-by-Step 3.18 demonstrates how to invoke a remote object hosted in the IIS.

**STEP BY STEP 3.18: Instantiating and Invoking an IIS-Hosted Remote Object**

1. Add a new Visual C# .NET Windows application named StepByStep3\_18 to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting and the project StepByStep3\_9 (the interface assembly).
3. In the Solution Explorer, rename the default Form1.cs to DbConnectClient.cs. Open the form in code view and change all occurrences of Form1 to refer to DbConnectClient instead.
4. Add the following using directives:
5. using System.Runtime.Remoting;
6. using System.Runtime.Remoting.Channels;
7. using System.Runtime.Remoting.Channels.Http;

using StepByStep3\_9; // contains IDbConnect

1. Place two GroupBox controls, a TextBox control (txtQuery, with its MultiLine property set to true), a Button control (btnExecute), and a DataGrid control (dgResults) on the form. Refer to Figure 3.5 for the design of this form.
2. Add the following code in the class definition:
3. // Declare a Remote object

IDbConnect dbc;

1. Double-click the form and add the following code in the Load event handler:
2. private void DbConnectClient\_Load(
3. object sender, System.EventArgs e)
4. {
5. // Register an HTTP client channel
6. HttpClientChannel channel =
7. new HttpClientChannel();
8. ChannelServices.RegisterChannel(channel);
9. // Instantiate the remote class
10. dbc = (IDbConnect)
11. Activator.GetObject(typeof(IDbConnect),
12. @"http://localhost/StepByStep3\_17/DbConnect.rem");

}

1. Double-click the Button control and add the following code in the Click event handler:
2. private void btnExecute\_Click
3. (object sender, System.EventArgs e)
4. {
5. try
6. { // Invoke a method on the remote object
7. this.dgResults.DataSource =
8. dbc.ExecuteQuery(this.txtQuery.Text);
9. dgResults.DataMember = "Results";
10. }
11. catch(Exception ex)
12. {
13. MessageBox.Show(ex.Message,
14. "Query Execution Error");
15. }

}

1. Build the project. This step creates the remoting client that can invoke an IIS-hosted remote object.
2. Set StepByStep3\_18, the remoting client, as the startup project. Select Debug, Start Without Debugging to run the project. Enter a query in the text box and click the button. The code invokes a method on the remote object, which is created when the form loads. The code binds the results from the remote method to the DataGrid control.

This client was similar to the client created in Step-by-Step 3.12; however, there are a few things to note. The client is using the HTTP channel to communicate with the server, there is no port number specified in the URL, and the URL ends with .rem.

**Asynchronous Remoting**

**Create and consume a .NET Remoting object: Implement an asynchronous method.**

So far, all the method invocations that you've studied in this chapter have been synchronous. In a synchronous method call, the thread that executes the method call waits until the called method finishes execution.

**NOTE**

**Asynchronous Method Calls** For asynchronous method calls, the remote objects need not explicitly support the asynchronous behavior. It is up to the caller to decide whether a particular remote call is asynchronous or not.

Such types of synchronous calls can make the user interface very non-responsive if the client program is waiting for a long process to finish executing on the server. This is especially true for remote method calls where additional time is involved because the calls are made across the network.

The .NET framework has a solution for this in the form of asynchronous methods. An asynchronous method calls the method and returns immediately, leaving the invoked method to complete its execution.

**Understanding the Model of Asynchronous Programming in the .NET Framework**

In the .NET Framework, asynchronous programming is implemented with the help of delegate types. Delegates are types that are capable of storing references to methods of a specific signature. A delegate declared as follows is capable of invoking methods that take a string parameter and return a string type:

delegate string LongProcessDelegate(string param);

So if there is a method definition, such as the following:

string LongProcess(string param)

{

...

}

then the LongProcessDelegate can hold references to this method like this:

LongProcessDelegate delLongProcess;

delLongProcess = new LongProcessDelegate(LongProcess);

After you have the delegate object available, you can call its BeginInvoke() method to call the LongProcess() method asynchronously, such as in the following:

IAsyncResult ar =

delLongProcess.BeginInvoke("Test", null, null);

Here the IAsyncResult interface is used to monitor an asynchronous call and relate the beginning and the end of an asynchronous method call. When you use the BeginInvoke() method, the control immediately comes back to the next statement while the LongProcess() method may still be executing.

To return the value of an asynchronous method call, you can call the EndInvoke() method on the same delegate, such as in the following:

String result = delLongProcess.EndInvoke(ar);

However, it is important to know where to place this method call, because when you call the EndInvoke() method, if the LongProcess() method has not yet completed execution, the EndInvoke() method causes the current thread to wait for the completion of the LongProcess() method. A poor use of the EndInvoke() method, such as placing it in just the next statement after the BeginInvoke() method, can cause an asynchronous method call to result in a synchronous method call.

One of the alternatives to this problem is to use the IsCompleted property of the IAsyncResult object to check whether the method has completed the execution, and then call the EndInvoke() method only in such a case.

string result;

if(ar.IsCompleted)

{

result = delLongProcess.EndInvoke(ar);

}

But regular polling of the IAsyncResult.IsCompleted property requires additional work at the client side.

You can also use a WaitHandle object to manage asynchronous remote method calls. When you are ready to wait for the results of the remote method call, you can retrieve a WaitHandle object from the IAsyncResult object's AsyncWaitHandle property:

ar.AsyncWaitHandle.WaitOne();

String result = delLongProcess.EndInvoke(ar);

The WaitOne() method of the WaitHandle object causes the thread to pause until the results are ready. The WaitHandle object has other methods that are useful if you have multiple outstanding asynchronous remote method calls. You can wait for all the method calls to come back by using the static WaitHandle.WaitAll() method, or for the first one to return by using the static WaitHandle.WaitAny() method. Therefore, the WaitHandle object essentially lets you turn the asynchronous process back into a synchronous process.

There is, in fact, a better technique for managing asynchronous method invocation—callback methods. In this technique, you can register a method that is automatically invoked as soon as the remote method finishes execution. You can then place a call to the EndInvoke() method inside the callback method to collect the result of remote method execution.

To implement the callback technique with an asynchronous method invocation, you need to take the following steps in the client program:

1. Define a callback method that you want to execute when the remote method has finished execution.
2. Create an object of delegate type AsyncCallback to store the reference to the method created in step 1.
3. Create an instance of the remote object on which you wish to invoke remote method calls.
4. Declare a delegate type capable of storing references to the remote method.
5. Using the object in step 3, create a new instance of delegate declared in step 4 to refer to the remote method.
6. Call the BeginInvoke() method on the delegate created in step 5, passing any arguments and the AsyncCallback object.
7. Wait for the server object to call your callback method when the method has completed.

**Applying Asynchronous Programming**

In the following sections I create a set of three projects to demonstrate the use of the callback method for an asynchronous method call:

* The Remotable Class—The remotable class exposes the remote methods that are called from the client program.
* The Remote Server—I use IIS to host the remotable object.
* The Client Program—The client program calls the remote method asynchronously. I use the Soapsuds tool to automatically generate the metadata required to invoke the remote object.

I create a remotable class that is different from other remotable classes used in this chapter to help you properly understand synchronous and asynchronous method calls.

The remotable class used in Step-by-Step 3.19 is named RemotableClass and it has a single method named LongProcess(). The LongProcess() method waits for about 5 seconds to simulate a long process call.

**STEP BY STEP 3.19: Creating a Remotable Class**

1. Add a new Visual C# .NET class library named StepByStep3\_19 to the solution.
2. In the Solution Explorer, rename the default Class1.cs to RemotableClass.cs.
3. Open the RemotableClass.cs and replace the code with the following code:
4. using System;
5. using System.Threading;
6. namespace StepByStep3\_19
7. {
8. public class RemotableClass : MarshalByRefObject
9. {
10. public string LongProcess(string param)
11. {
12. Thread.Sleep(5000);
13. return param;
14. }
15. }

}

1. Select Build, Build StepByStep3\_19. This step generates the code for your class library and packages it into the file StepByStep3\_19.dll, which is located in the bin\Debug or bin\Release directory of your project.

RemotableClass is now ready to be hosted in a remoting host. In Step-by-Step 3.20, I decided to use IIS to host the remotable class because it requires only a minimum amount of code. I control the remoting configuration by modifying the web.config file.

**STEP BY STEP 3.20: Using IIS to Host a Remotable Class**

1. Add a new Visual C# ASP.NET application named StepByStep3\_20 to the solution.
2. Add references the project StepByStep3\_19 (the remotable class assembly).
3. Open the web.config file and add the following <system.runtime.remoting> element inside the <configuration> element:
4. <configuration>
5. <system.runtime.remoting>
6. <application>
7. <service>
8. <!-- Set the activation mode,
9. remotable object, and its URL -->
10. <wellknown mode="Singleton"
11. type="StepByStep3\_19.RemotableClass, StepByStep3\_19"
12. objectUri="RemotableClass.rem" />
13. </service>
14. </application>
15. </system.runtime.remoting>
16. ...

</configuration>

1. The remoting server is now hosting the RemotableClass contained in the assembly StepByStep3\_19 for remote invocation by using the Singleton activation mode.

Both RemotableClass, the remote object and the remoting host contain no additional information that supports an asynchronous method call. The asynchronous call is completely dependent on the client code. So in the final step, I create a client application that calls a remote method synchronously as well as asynchronously. Seeing both ways of calling a method will help you note the difference.

**STEP BY STEP 3.21: Instantiating and Invoking a Client That Calls Remote Object Methods Synchronously and Asynchronously**

1. Add a new Visual C# .NET Windows application named StepByStep3\_21 to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting.
3. Select Start, Programs, Microsoft Visual Studio .NET, Visual Studio .NET Tools, Visual Studio .NET Command Prompt to launch a .NET command prompt.
4. Navigate to the StepByStep3\_13 project and run the following command to automatically generate the source code for the interface of the remotable class RemotableClass, registered by the IIS Web server:
5. soapsuds -nowp –gc
6. -url:http://localhost/StepByStep3\_20/RemotableClass. rem?wsdl

This step generates a file named StepByStep3\_19.cs, which contains the interface code for the RemotableClass class.

1. **5**In the Solution Explorer, right-click project StepByStep3\_21 and select Add, Add Existing Item to add the StepByStep3\_19.cs file (generated by the soapsuds.exe in step 4).
2. Open the file StepByStep3\_19.cs. You will notice that the RemotableClass class definition is applied with two attributes—Serializable and SoapType. Remove the SoapType attribute definition so that the class definition looks like this:
3. [Serializable]
4. public class RemotableClass :
5. System.MarshalByRefObject
6. {
7. ...

}

1. In the Solution Explorer, rename the default Form1.cs to SyncAsync.cs. Open the form in code view and change all occurrences of Form1 to refer to SyncAsync instead.
2. Add the following using directives:
3. using System.Runtime.Remoting;
4. using System.Runtime.Remoting.Channels;
5. using System.Runtime.Remoting.Channels.Http;
6. // Contains RemotableClass interface

using StepByStep3\_19;

1. Place a Label control, a TextBox control (txtResults), and two Button controls (btnSync and btnAsync) on the form. Refer to Figure 3.12 for the design of this form.
2. Add the following code to the class definition:
3. // Remotable object
4. RemotableClass remObject;
5. // Create a delegate for the LongProcess method
6. // of the Remotable object
7. delegate string LongProcessDelegate(string param);
8. // Declare a LongProcessDelegate object,
9. // an AsyncCallback delegate object,
10. // and an IAsyncResult object
11. LongProcessDelegate delLongProcess;
12. AsyncCallback ab;
13. IAsyncResult ar;
14. // Declare a static integer variable to hold
15. // the number of times the method is called

static int counter;

1. Double-click the form and add the following code in the Load event handler:
2. private void SyncAsync\_Load(
3. object sender, System.EventArgs e)
4. {
5. // Register an HTTP client channel
6. HttpClientChannel channel =
7. new HttpClientChannel();
8. ChannelServices.RegisterChannel(channel);
9. // Instantiate the remote class
10. remObject = (RemotableClass) Activator.GetObject(
11. typeof(RemotableClass),
12. @"http://localhost/StepByStep3\_20/RemotableClass.rem");
13. // Create an AsyncCallback delegate object to hold
14. // the reference of the LongProcessCompleted
15. // callback method, which is called when
16. // the asynchronous call is completed
17. ab = new AsyncCallback(LongProcessCompleted);
18. // Create a LongProcessDelegate delegate object to
19. // hold the reference of the LongProcess method
20. delLongProcess = new LongProcessDelegate(
21. remObject.LongProcess);

}

1. Double-click the btnSync control and add the following code in the Click event handler:
2. private void btnSync\_Click(
3. object sender, System.EventArgs e)
4. {
5. // Increment the method call counter
6. counter++;
7. string param = String.Format(
8. "Call: {0}, Type=Synchronous", counter);
9. // Append the start message to the text box
10. txtResults.AppendText(String.Format(
11. "{0}, Started at: {1}\n",
12. param, DateTime.Now.ToLongTimeString()));
13. // Call the LongProcess method of the
14. // remotable object
15. remObject.LongProcess(param);
16. // Append the completed message to the text box
17. txtResults.AppendText(
18. String.Format("{0}, Completed at: {1}\n",
19. param, DateTime.Now.ToLongTimeString()));

}

1. Double-click the btnAsync control and add the following code in the Click event handler:
2. private void btnAsync\_Click(
3. object sender, System.EventArgs e)
4. {
5. // Increment the method call counter
6. counter++;
7. string param = String.Format(
8. "Call: {0}, Type=Asynchronous",
9. counter);
10. // Append the start message to the text box
11. txtResults.AppendText(
12. String.Format("{0}, Started at: {1}\n",
13. param, DateTime.Now.ToLongTimeString()));
14. // Call the BeginInvoke method to start
15. // the asynchronous method call
16. ar = delLongProcess.BeginInvoke(param, ab, null);

}

1. Add the following callback method definition for LongProcessCompleted() to the class:
2. void LongProcessCompleted(IAsyncResult ar)
3. {
4. // Call the EndInvoke method to retrieve the
5. // return value of the asynchronous method call
6. string result = delLongProcess.EndInvoke(ar);
7. // Append the completed message to the text box
8. txtResults.AppendText(
9. String.Format("{0}, Completed at: {1}\n",
10. result, DateTime.Now.ToLongTimeString()));

}

1. Build the project. This step creates a remoting client for the RemotableClass remotable object.
2. Set StepByStep3\_21, the remoting client, as the startup project. Select Debug, Start Without Debugging to run the project. Click the Call Long Process Synchronously button to call the LongProcess() method synchronously. You should see the start message appended to the text box, and the application freezes for the next five seconds. After the method call is completed you notice a completed message and you are now able to work with the application. Now click the Call Long Process Asynchronously button to call the LongProcess() method asynchronously. As soon as the method starts, the start message is appended to the text box. While the method is executing, you can still work with the application (such as by moving the form, clicking buttons, and so on, although clicking the synchronous call button freezes the application). When the method is completed, you notice a completed message in the text box, as shown in [Figure 3.12](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig12.jpg')).

[**[Figure 3.12](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig12.jpg'))Figure 3.12**](javascript:popUp('/content/images/chap3_0789728249/elementLinks/03fig12.jpg')) **The asynchronous method call invokes the method and transfers the control back to the form. The asynchronous call can use a callback method to get a notification when the remote method finishes execution.**

In this program I am following the same steps for an asynchronous method call that were discussed earlier in the previous section, "Understanding the Model of Asynchronous Programming in the .NET Framework."

The only surprise in Step-by-Step 3.21 is the special use of the Soapsuds tool. Step 4 uses the Soapsuds tool to generate the source code of the interface assembly instead of the assembly itself. The soapsuds-generated assembly uses a SoapType attribute that doesn't work with delegates. The advantage of soapsuds-generated source code is that I can manually remove the problematic SoapType attribute and compile the program to generate the interface assembly.

**REVIEW BREAK**

* You can choose to run the remoting host as a console application, as a Windows service, or as an IIS application.
* You can use IIS as an activation agent only when the underlying communication is in the HTTP channel. Using IIS as an activation agent eliminates the need to write a separate server program that listens on a unique port number (IIS uses the port 80).
* When creating IIS-hosted remote objects, you cannot specify constructor parameters; therefore using IIS to activate CAO is not possible.
* You can invoke a method asynchronously by calling the BeginInvoke() method on that method's delegate.
* You can automatically get a notification when an asynchronous method ends. However, you must first create another delegate object (of AsyncCallback type) that refers to the callback method that you need to execute when the remote method ends. And then you should pass the delegate to the callback method as an argument to the BeginInvoke() method.

**Chapter Summary**

The .NET Framework is a modern platform for building distributed applications. In this chapter you learned how to create distributed applications by using the remoting library of the .NET Framework.

You learned that the remoting framework is made up of several elements, such as remotable classes, remoting hosts, remoting clients, channels, formatters, and so on. Each of these elements is configurable and extensible. You practiced creating a remotable class and hosting that class in several modes such as server-activated SingleCall, server-activated Singleton, and client-activated. You also used different channels, such as HTTP and TCP, and different formatters, such as SOAP and binary.

I also demonstrated the use of a configuration file to configure the remoting configuration for a remoting host and the remoting client. You noted that depending on your requirements, you can write remoting configurations at two level: one at the application level (.exe.config, or web.config for IIS) and the other at the machine level (machine.config).

Finally, I discussed the technique of invoking remote methods asynchronously. An asynchronous method call makes a user interface–based application, such as a Windows forms application, quite responsive to user input despite the delays caused in a remote method call.

**Key Terms**

* application domain
* asynchronous call
* channel
* marshaling
* proxy
* synchronous call
* remoting
* runtime host

**Apply Your Knowledge**

**Exercises**

**3.1 - Using HTTP Channels with Binary Formatters**

As I discussed in the chapter, the default formatter for the HTTP channel is SOAP, and for the TCP channel is Binary. The formatters are used for serializing and deserializing messages in the specified encoding.

The chapter has made use of these default formatters in the examples. In this exercise, you'll learn how to configure a channel to use a formatter different from the default one.

In particular I'll demonstrate how to use the binary formatter with the HTTP channel. This combination is especially useful when you want to optimize the performance of a remote object that is hosted on IIS. However, you should note that binary format is proprietary to the .NET Framework.

**Estimated Time:** 30 minutes.

1. Launch Visual Studio .NET. Select File, New, Blank Solution, and name the new solution 320C03Exercises. Click OK.
2. Add a new empty Web project named Exercise3\_1\_Server to the solution.
3. Add references to StepByStep3\_9.dll (the interface assembly containing IDbConnect) and StepByStep3\_10.dll (the remotable object, DbConnect).
4. Add a new Web configuration file to the project. Open the web.config file and add the following <system.runtime.remoting> element inside the <configuration> element:
5. <configuration>
6. <system.runtime.remoting>
7. <application>
8. <service>
9. <!--Set the activation mode,
10. remotable object,
11. and its URL -->
12. <wellknown mode="Singleton"
13. type=
14. "StepByStep3\_10.DbConnect, StepByStep3\_10"
15. objectUri=
16. "DbConnect.rem" />
17. </service>
18. </application>
19. </system.runtime.remoting>
20. ...

</configuration>

IIS is now hosting the StepByStep3\_10.DbConnect remotable class as a server-activated object by using the Singleton activation mode.

1. Add a new Visual C# .NET Windows application named Exercise3\_1\_Client to the solution.
2. Add references to StepByStep3\_9.dll (the interface assembly containing IDbConnect).
3. In the Solution Explorer, right-click project Exercise3\_1\_Client and select Add, Add New Item from the context menu. Add an item named Exercise3\_1\_Client.exe.config, based on the XML file template.
4. Open the Exercise3\_1\_Client.exe.config file and modify it to contain the following code:

<configuration>

<system.runtime.remoting>

<application>

<channels>

<channel ref="http">

<serverProviders>

<formatter ref =

"binary" />

</serverProviders>

</channel>

</channels>

</application>

</system.runtime.remoting>

</configuration>

1. In the Solution Explorer, select the project and click the Show All Files button in the tool bar. Move the Exercise3\_1\_Client.exe.config file from the project folder to the bin\Debug folder under the project, where the Exercise3\_1\_Client.exe file will be created when the project is compiled.
2. In the Solution Explorer, rename the default Form1.cs to DbConnectClient.cs. Open the form in code view and change all occurrences of Form1 to refer to DbConnectClient instead.
3. Place two GroupBox controls (grpQuery and grpResults), a TextBox control (txtQuery, with its MultiLine property set to true), a Button control (btnExecute), and a DataGrid control (dgResults) on the form. Refer to Figure 3.5 for the design of this form.
4. Add the following using directives:

using System.Runtime.Remoting;

using StepByStep3\_9;

1. Add the following code in the class definition:

// Declare a Remote object

IDbConnect dbc;

1. Double-click the form and add the following code in the Load event handler:

private void DbConnectClient\_Load(

object sender, System.EventArgs e)

{

// Load remoting configuration

RemotingConfiguration.Configure

("Exercise3\_1\_Client.exe.config");

// Instantiate the remote class

dbc = (IDbConnect)

Activator.GetObject(typeof(IDbConnect),

@"http://localhost/Exercise3\_1\_Server/DbConnect.rem");

}

1. Double-click the Button control and add the following code in the Click event handler:

private void btnExecute\_Click(

object sender, System.EventArgs e)

{

try

{

// Invoke a method on

// the remote object

this.dgResults.DataSource =

dbc.ExecuteQuery(

this.txtQuery.Text);

dgResults.DataMember = "Results";

}

catch(Exception ex)

{

MessageBox.Show(ex.Message,

"Query Execution Error");

}

}

1. Build the project. Set Exercise3\_1\_Client, the remoting client, as the startup project. Select Debug, Start Without Debugging to run the project. Enter a query in the text box and click the button. The code invokes a method on the remote object. The remote object is serialized and deserialized in binary format and is transported over the HTTP. The code binds the results from the remote method to the DataGrid control.

Note that you can specify the desired formatter for the client and need not specify the formatter for the server. The formatter requested by the client will be used to format data by the server for that client.

**3.2 - Dynamically Publishing a Well-Known Object**

Well-known objects cannot be invoked from a client with a non-default constructor. However, you can create an object using any constructor you wish, initialize it any way you wish, and then make it available to clients.

You should use the RemotingServices.Marshal() method to publish an existing object instance.

**Estimated Time:** 30 minutes.

1. Add a new Visual C# console application named Exercise3\_2\_Server to the solution.
2. Add references to the .NET assembly System.Runtime.Remoting, the StepByStep3\_9.dll (the interface assembly containing IDbConnect), and StepByStep3\_10.dll (the remotable object, DbConnect).
3. In the Solution Explorer, rename the default Class1.cs to DbConnectServer.cs. Open the file and change the name of the class to DbConnectServer in the class declaration.
4. Add the following using directives:

using System.Runtime.Remoting;

using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Tcp;

using StepByStep3\_10;

1. Add the following code in the Main() method:

[STAThread]

static void Main(string[] args)

{

// Create and register a TCP server

// channel that listens on port 1234

TcpServerChannel channel =

new TcpServerChannel(1234);

ChannelServices.RegisterChannel(

channel);

// Create the remotable object here

// itself. Call the

// RemotingServices.Marshal() method

// to marshal (serialize) the created

// remotable object to transfer the

// object beyond application boundaries

// with the specified uri

DbConnect dbcPubs =

new DbConnect("Pubs");

RemotingServices.Marshal(

dbcPubs, "Pubs.uri");

DbConnect dbcNorthwind =

new DbConnect("Northwind");

RemotingServices.Marshal(

dbcNorthwind, "Northwind.uri");

Console.WriteLine(

"Started server in the " +

"Singleton mode");

Console.WriteLine(

"Press <ENTER> to terminate " +

"server...");

Console.ReadLine();

}

1. Build the project. This step creates a remoting server that creates the remotable object StepByStep3\_1.DbConnect and is capable of marshaling the remote object across application boundaries.
2. Add a new Visual C# .NET Windows application named Exercise3\_2\_Client to the solution.
3. Add references to the .NET assembly System.Runtime.Remoting and the project StepByStep3\_9 (the interface assembly).
4. In the Solution Explorer, rename the default Form1.cs to DbConnectClient.cs. Open the form in code view and change all occurrences of Form1 to refer to DbConnectClient instead.
5. Add the following using directives:

using System.Runtime.Remoting;

using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Tcp;

using StepByStep3\_9;

1. Place three GroupBox controls (grpDatabases, grpQuery, and grpResults), a ComboBox control (cboDatabases), a TextBox control (txtQuery), a Button control (btnExecute), and a DataGrid control (dgResults) on the form. Set the DropDownStyle to DropDownList.
2. Select the Items property of the cboDatabases control in the Properties window and click on the (...) button. This opens the String Collection Editor dialog box. Enter the following database names in the editor:

Northwind

Pubs

1. Click OK to add the databases to the Items collection of the cboDatabases control.
2. Add the following code in the class definition:

// Declare a remote object

IDbConnect dbc;

1. Double-click the form and add the following code in the Load event handler:

private void DbConnectClient\_Load(

object sender, System.EventArgs e)

{

// Register a TCP client channel

TcpClientChannel channel =

new TcpClientChannel();

ChannelServices.RegisterChannel(

channel);

cboDatabases.SelectedIndex = 0;

}

1. Double-click the cboDatabases control and add the following code in the SelectedIndexChanged event handler:

private void

cboDatabases\_SelectedIndexChanged(

object sender, System.EventArgs e)

{

switch (

cboDatabases.SelectedItem.ToString())

{

case "Pubs":

{

// Instantiate the remote class

dbc = (IDbConnect)

Activator.GetObject(

typeof(IDbConnect),

@"tcp://localhost:1234/Pubs.uri");

break;

}

case "Northwind":

{

// Instantiate the remote class

dbc = (IDbConnect)

Activator.GetObject(

typeof(IDbConnect),

@"tcp://localhost:1234/Northwind.uri");

break;

}

}

}

1. Double-click the btnExecute control and add the following code in the Click event handler:

private void btnExecute\_Click

(object sender, System.EventArgs e)

{

try

{

// Invoke a method on

// the remote object

this.dgResults.DataSource =

dbc.ExecuteQuery(

this.txtQuery.Text);

dgResults.DataMember = "Results";

}

catch(Exception ex)

{

MessageBox.Show(ex.Message,

"Query Execution Error");

}

}

1. Build the project. You now have a remoting client ready to use.
2. Set the Exercise3\_2\_Server, the remoting server, as the startup project. Select Debug, Start Without Debugging to run the project. You should see a command window displaying a message that the server is started in the singleton activation mode. You should also see messages that the remote object is already created.
3. Set Exercise3\_2\_Client, the remoting client, as the startup project. Select Debug, Start Without Debugging to run the project. Select a desired database from the combo box, enter a query in the text box, and click the button. The code invokes a method on the remote object that is already created on the server. The code binds the results from the remote method to the DataGrid control.
4. Again select Debug, Start Without Debugging to run one more instance of the remoting client. Select a different database from the combo box, enter a query in the text box, and click the button. You should see that the second instance of the client fetches the data from the selected database.

**Review Questions**

1. What is an application domain? How does the CLR manage an application domain?
2. What are MBR objects? What are their advantages and disadvantages?
3. What do you mean by a channel? What are the different types of channels the .NET Framework provides?
4. What are the advantages and disadvantanges of the binary and SOAP formatters?
5. What are the two modes for creating Server-Activated objects?
6. When should you choose to create a client- activated object?
7. What is the benefit of using declarative configuration over programmatic configuration?
8. What are the two methods of the Activator class that enable you to create instances of remote objects?
9. What are the advantages of using an IIS server as an activation agent?
10. What should you do while creating a remotable class so that its methods can be called asynchronously?

**Exam Questions**

1. You are designing a distributed application that hosts a remote object. You want only the authorized client applications to activate the remote object. You want to write the application with the minimum amount of code. Which of the following channels enables you to achieve this objective? (Select two.)
   1. HttpChannel
   2. HttpServerChannel
   3. TcpChannel
   4. TcpServerChannel
2. You are designing a company-wide order processing system. This application is hosted on a server in the company's headquarters in Redmond, Washington, and is accessed by 1500 franchise locations throughout the world. The application specification mentions that the franchisees should be able to access the order-processing system even through firewalls. A large number of franchisees access the application over a slow connection and your objective is to maximize the performance of the application. Which of the following combinations of channel and formatter would you choose in this scenario?
   1. Use a TCP channel with a binary formatter.
   2. Use a TCP channel with a SOAP formatter.
   3. Use an HTTP channel with a binary formatter.
   4. Use an HTTP channel with a SOAP formatter.
3. You are designing a distributed application for a large automotive company. The application allows the part suppliers across the globe to collect the latest design specifications for a part. The application is heavily accessed by the suppliers. For greater scalability, you are required to design the application so that it can be deployed in a load-balanced environment. How should you host the remotable object in this scenario?
   1. As a server-activated object in SingleCall activation mode.
   2. As a server-activated object in Singleton activation mode.
   3. As a client-activated object using the HTTP channel.
   4. As a client-activated object using the SOAP formatter.
4. You have been hired by Great Widgets Inc., to create an application that enables their suppliers to access the purchase order information in real time. You create the required classes that the suppliers can activate remotely and package them into a file named gwpoinfo.dll. You plan to use IIS as the remoting host for this file. After the application has been deployed, minimal steps should be involved in changing the remoting configuration for this application. Any config-uration changes made to the purchase order information system should not affect any other applications running on that server. Which of the following files would you choose to configure remoting for the purchase order information system?
   1. gwpoinfo.dll
   2. web.config
   3. global.asax
   4. machine.config
5. You have designed a remotable class named ProductDesign. You now want to register this class with the remoting system in such a way that client programs should be able to remotely instantiate objects of this class and invoke methods on it. You want to have only one instance of this class on the server, no matter how many clients connect to it. Which of the following code segments fulfills your requirement?
   1. RemotingConfiguration.
   2. RegisterWellKnownServiceType(
   3. typeof(ProductDesign),
   4. "ProductDesign",
   5. WellKnownObjectMode.SingleCall
   6. );
   7. RemotingConfiguration.
   8. RegisterWellKnownServiceType(
   9. typeof(ProductDesign),
   10. "ProductDesign",
   11. WellKnownObjectMode.Singleton
   12. );
   13. RemotingConfiguration.
   14. RegisterActivatedServiceType(
   15. typeof(ProductDesign),
   16. "ProductDesign"
   17. );
   18. RemotingConfiguration.
   19. RegisterWellKnownClientType(
   20. typeof(ProductDesign),
   21. "ProductDesign"
   22. );
6. You have designed a remotable class that allows the user to retrieve the latest weather information for their region. You do not want to write a lot of code to create a custom remoting host, so you decide to use IIS to host the application. The name of the remotable class is RemotingWeather.WeatherInfo and it is stored in an assembly named WeatherInfo.dll. You want the users to access this remotable class by using the URL http://RemoteWeather.com/users/WeatherInfo.rem. Which of the following remoting configurations would you place in the web.config file so that client applications can correctly retrieve weather information?
   1. <system.runtime.remoting>
   2. <application>
   3. <service>
   4. <activated type=
   5. "RemotingWeather.WeatherInfo, WeatherInfo"
   6. />
   7. </service>
   8. <channels>
   9. <channel ref="http" port="80" />
   10. </channels>
   11. </application>
   12. </system.runtime.remoting>
   13. <system.runtime.remoting>
   14. <application>
   15. <service>
   16. <wellknown mode="Singleton"
   17. type=
   18. "RemotingWeather.WeatherInfo, WeatherInfo"
   19. objectUri="WeatherInfo.rem"
   20. />
   21. </service>
   22. </application>
   23. </system.runtime.remoting>
   24. <system.runtime.remoting>
   25. <application>
   26. <service>
   27. <activated type=
   28. "RemotingWeather.WeatherInfo, WeatherInfo"
   29. />
   30. </service>
   31. <channels>
   32. <channel ref="http server"
   33. port="80" />
   34. </channels>
   35. </application>
   36. </system.runtime.remoting>
   37. <system.runtime.remoting>
   38. <application>
   39. <client>
   40. <wellknown mode="Singleton"
   41. type=
   42. "RemotingWeather.WeatherInfo, WeatherInfo"
   43. objectUri="WeatherInfo.rem"
   44. />
   45. </client>
   46. </application>
   47. </system.runtime.remoting>
7. You are a software developer for LubriSol, Inc., which manufactures chemicals for automobile industries. Your company does a lot of business with ReverseGear, Inc., which is the largest manufacturer of heavy vehicles in the country. ReverseGear, Inc., uses a .NET remoting application that allows its suppliers to check its daily parts requirements. Your objective is to create a client application to the ReverseGear's application that retrieves the information for parts produced by your company. All you know about the server application is its URL, which is http://ReverseGearInc.com/Suppliers/Req.rem. You want the quickest solution. What should you do to write a client application successfully?
   1. Contact ReverseGear, Inc., to ask for the interface and include reference to the interface in the client project.
   2. Open the URL in the Web browser and select View, Source to find how the remote class is structured.
   3. Use the Visual Studio .NET Add Web Reference feature to add a reference to the remote class in the client project.
   4. Use the Soapsuds tool to automatically generate the metadata and include the reference to this metadata in the client project.
8. You want to host a remotable class via the .NET remoting framework so that remote clients can instantiate the class and invoke methods on it. The remotable class does not have any user interface but it must use Integrated Windows authentication to authenticate the users. Which of the following techniques you should use to host the remotable class? You want a solution that requires you to write minimal code.
   1. Use a console application as a remoting host.
   2. Create a Windows service and use that to host the remotable class.
   3. Use a Windows forms application to host the remotable class.
   4. Use Internet Information Services (IIS) as a remoting host.
9. You are developing a remoting client to access a server-activated remotable object hosted at the URL tcp://finance:1234/Budget. You have obtained an interface assembly of this remote object. This assembly contains an interface named IBudget that is implemented by the remote class. You want to instantiate the remote object to invoke a method name GetDepartmentBudget(), which accepts a string value and returns a double value containing the department budget. Given the following code, what should you write in line 06 to successfully invoke the GetDepartmentBudget() method?
10. 01: // Register a TCP client channel
11. 02: TcpClientChannel channel =
12. new TcpClientChannel();
13. 03: ChannelServices.RegisterChannel(
14. channel);
15. 04: IBudget budget;
16. 05: // Instantiate the remote class
17. 06:
18. 07: // Invoke the remote method
19. 08: double budgetValue =

budget.GetDepartmentBudget("HR");

* 1. budget = (IBudget)
  2. Activator.GetObject(typeof(IBudget),
  3. @"tcp://finance:1234/Budget");
  4. budget = (IBudget)
  5. Activator.CreateInstance(
  6. typeof(IBudget),
  7. @"tcp://finance:1234/Budget");
  8. budget = new IBudget();
  9. RemotingConfiguration.
  10. RegisterWellKnownClientType(
  11. typeof(IBudget),
  12. @"tcp://finance:1234/Budget");
  13. Budget = new IBudget();

1. You are developing an application that allows the client programs to instantiate a class named Inventory. You want the remote object to be created on the server so that it can access the inventory database. However, you want client programs to control the creation and the lifetime of the remote objects. Which of the following methods of the RemotingConfiguration class would you choose to register the remotable class with the remoting system on the server?
   1. RegisterWellKnownServiceType()
   2. RegisterWellKnownClientType()
   3. RegisterActivatedServiceType()
   4. RegisterActivatedClientType()
2. You work for a large chemical manufacturing company that has four production units across the country. Your team has the responsibility of designing a distributed application that allows different production units to share and update material safety information for various products. One of your co-workers is using the following code to create a remoting host to host a server-activated object. She is getting an error. What should she do to resolve this error?
3. 01: using System.Runtime.Remoting;
4. 02: using System.Runtime.Remoting.Channels;
5. 03: using System.Runtime.Remoting.Channels.Tcp;
6. 04: using
7. System.Runtime.Remoting.Channels.Http;
8. 05: [STAThread]
9. 06: static void Main(string[] args)
10. 07: {
11. 08: TcpServerChannel tcpChannel =
12. new TcpServerChannel(7777);
13. 09: HttpServerChannel httpChannel =
14. new HttpServerChannel(8888);
15. 10: RemotingConfiguration.
16. RegisterWellKnownServiceType
17. (typeof(MsdsInfo), "MsdsInfo",
18. WellKnownObjectMode.Singleton);

11: }

* 1. Remove the statement at line 09.
  2. Add the following statements just before line 10:

ChannelServices.RegisterChannel(tcpChannel);  
ChannelServices.RegisterChannel(httpChannel);

* 1. In the statement at line 08, replace TcpServerChannel with TcpChannel, and similarly in the statement in line 09, replace HttpServerChannel with HttpChannel.
  2. Use the same port numbers in the statements in line 08 and line 09.

1. One of your coworkers has written the following code as part of a client application that activates a remote object. She is complaining that her program is not compiling. What should she modify in the program to remove this error? (Select all that apply.)
2. 01: DbConnect CreateObject()
3. 02: {
4. 03: TcpClientChannel channel =
5. new TcpClientChannel(1234);
6. 04: ChannelServices.RegisterChannel(
7. channel);
8. 05: RemotingConfiguration.
9. RegisterWellKnownClientType(
10. 06: typeof(DbConnect),
11. 07: "tcp://localhost/DbConnect"
12. 08: );
13. 09: dbc = new DbConnect();
14. 10: return dbc;

11: }

* 1. Change line 05 to use the RegisterWellKnownServiceType() method instead of the RegisterWellKnownClientType() method.
  2. Change the URL in line 07 to "tcp:// localhost:1234/DbConnect"
  3. Remove the port number from the constructor of TcpClientChannel() in line 03
  4. Change the code in line 07 to objectUri="DbConnect"

1. The Soapsuds tool (soapsuds.exe) can be used to automatically generate the interface assembly for the remotable object. Which of the following statements are FALSE related to the Soapsuds tool? (Select two options.)
   1. The Soapsuds tool can be used to generate metadata for server-activated objects.
   2. The Soapsuds tool can be used to generate metadata for client-activated objects.
   3. The Soapsuds tool can be used to generate metadata for remotable objects registered through the HTTP channel.
   4. The Soapsuds tool can be used to generate metadata for remotable objects registered through the TCP channel.
2. You have designed a Windows application that is used by the shipping department of a large distribution house. The Windows application instantiates a remotable class hosted on Internet Information Services (IIS). The remotable class provides various services to the Windows application, such as address validation and calculation of shipping rates. When you deploy the application, users complain that when they click the Validate Address button, the Windows application freezes and they can't take further action till the address has been verified. What should you do to improve the application's responsiveness?
   1. Use the binary formatter instead of the SOAP formatter.
   2. Use the TCP channel to communicate instead of the HTTP channel.
   3. Modify the remotable class to support asynchronous method calls.
   4. Modify the Windows application to call the methods asynchronously on the remote object.
3. When you derive a class from the MarshalByRefObject to make the class remotable, which of the following members of the class are not remoted? (Select all that apply.)
   1. non-static public methods
   2. static methods
   3. non-static private methods
   4. non-static public properties

**Answers to Review Questions**

1. The application domain, represented by the AppDomain class, is the basic unit of isolation for running applications in the CLR. The CLR allows several application domains to run within a single Windows process. The CLR ensures that code running in one application domain cannot affect other application domains. The CLR can terminate an application domain without stopping the entire process.
2. MBR objects are remotable objects that derive from the System.MarshalByRefObject class. The MBR objects always reside on the server; the client application domain holds only a reference to the MBR objects and uses a proxy object to interact with the MBR objects. They are best suited when the remotable objects are large or when the functionality of the remotable objects is available only in the server environment on which it is created. However, they increase the number of network roundtrips between the server application domain and the client application domain.
3. A channel is an object that transports messages across remoting boundaries such as application domains, processes, and computers. The .NET Framework provides implementations for HTTP and TCP channels.
4. The binary formatter represents messages in a compact and efficient way, whereas the SOAP formatters are very verbose. The SOAP formatters can be used to communicate in heterogeneous environments, whereas the binary formatters can be understood only by .NET applications.
5. The two modes for creating SAOs are SingleCall and Singleton. SingleCall activation mode creates a remote object for each client request. The object is discarded as soon as the request completes. Singleton activation mode creates a remote object only once and all the clients share the same copy. Hence SingleCall SAOs are stateless and Singleton SAOs maintain state global to all clients.
6. CAOs are created for each client whenever the client requests. Hence CAOs are best suited when clients want to maintain private sessions with remote objects, when the clients want to control the lifetimes of the objects, or when they want to create customized remote objects with non-default properties.
7. The main benefit of using declarative configuration over programmatic configuration is that you need not recompile the application after changing the remoting settings in the configuration file. The changes are picked up automatically.
8. The GetObject() method (for server-activated objects) and CreateInstance() method (for client-activated objects) are the two methods of the Activator class that enable you to create instances of remote objects.
9. Using IIS as an activation agent offers the following advantages:
   * You need not write a separate server program to register the remotable classes.
   * You need not worry about finding an available port for your server application. You can just host the remotable object and IIS automatically uses port 80.
   * IIS can provide other functionality, such as authentication and Secure Sockets Layer (SSL).
10. A remotable object by default is capable of being called asynchronously; therefore no extra efforts are required to create remote objects that can be called asynchronously.

**Answers to Exam Questions**

1. **A, B.** Your objective is to provide access to only authorized clients. Authorization is a function of the remoting host. IIS is the only available remoting host that provides you with this capability. IIS supports only HTTP communication. Therefore, you can use either the HttpChannel or HttpServerChannel channel because both allow you to listen to incoming messages from clients. IIS does not support TcpChannel and TcpServerChannel, so if you use these channels you have to write additional code to implement security and this is not desired in the given scenario. For more information, see the section "Choosing Between the HTTP and the TCP Channels" in this chapter.
2. **C.** Firewalls generally allow HTTP messages to pass through, and the binary formatter provides a size-optimized format for encoding data. Using TCP may require administrators to open additional ports in the firewall, and the SOAP format is verbose when compared to binary and would take additional bandwidth, which is not a desirable solution for clients using slow connections. For more information, see the section "Choosing Between the HTTP and the TCP Channels" and "Channels and Formatters" in this chapter.
3. **A.** Only server-activated objects in SingleCall activation mode support load-balancing because they do not maintain state across the method calls. For more information, see the section "Server-Activated Objects" in this chapter.
4. **B.** You should store the remoting configuration in the web.config file. This file is an XML-based configuration file that is easy to modify and does not require a separate compilation step. Storing configuration settings in gwpoinfo.dll or global.asax requires the additional step of compilation before the settings come into effect. The machine.config file is not suggested because any changes made to it affect all the applications running on the server. For more information, see the section "Using Configuration Files to Configure the Remoting Framework" in this chapter.
5. **B.** When you want to create just one instance of a remote object, without regard to the number of clients, you must create a server-activated object in the Singleton activation mode. For more information, see the section "Server-Activated Objects" in this chapter.
6. **B.** IIS supports only well-known or server- activated objects. Therefore, you must use the <wellknown> element rather than the <activated> element. Also, you are specifying the configuration for the server, so you must use the <server> element rather than the <client> element inside the <application> element to configure the WellKnown object. For more information, see the section "Using Configuration Files to Configure the Remoting Framework" and "Using IIS As an Activation Agent" in this chapter.
7. **D.** Because you know that the server's .NET remoting application is using HTTP, you can use the Soapsuds tool to automatically generate the metadata for the server. For more information, see the section "Using the Soapsuds Tool to Automatically Generate an Interface Assembly" in this chapter.
8. **D.** You should use IIS as the remoting host because IIS has built-in support for Integrated Windows authentication. You'll have to write additional code to achieve this with other techniques. For more information, see the section "Using IIS As an Activation Agent" in this chapter.
9. **A.** When you have an interface to a class and not the original class, you cannot use the new operator to instantiate the remote object. You should instead use the static methods of the Activator class. The Activator.GetObject() method is used to instantiate a server-activated object. For more information, see the section "Using Interface Assemblies to Compile Remoting Clients" in this chapter.
10. **C.** Your requirement is to register a client-activated remote object on the server, so you'll use the RegisterActivatedServiceType() method. The RegisterActivatedClientType() method is used to register the CAO with the remoting system in the client's application domain. The other two options are for creating the server-activated objects. For more information, see the section "Creating a Client-Activated Object" in this chapter.
11. **B.** In this program, although you have created an instance of TcpServerChannel and HttpServerChannel objects, you haven't yet registered them with the remoting framework. You'll register the channels by using the RegisterChannel() method of the ChannelServices class. For more information, see the sections "Using the SingleCall Activation Mode to Register a Remotable Class As a Server-Activated Object," "Using the Singleton Activation Mode to Register a Remotable Class As a Server-Activated Object," and "Registering a Remotable Class As a Client-Activated Object" in this chapter.
12. **B, C.** When creating a client channel you should not specify a port number when calling the channel constructor. Instead, the port number should be used with the URL of the remote object. After the channels are created they should be registered with the remoting system. For more information, see the section "Channels" in this chapter.
13. **B, D.** The Soapsuds tool is capable of generating metadata for server-activated objects on the HTTP channel. For more information, see the section "Using the Soapsuds Tool to Automatically Generate an Interface Assembly" in this chapter.
14. **D.** The issue in the question is not of speed but of responsiveness. This behavior occurs because the Windows application is calling the methods on the remote object synchronously. You can make the user interface more responsive by simply calling the remote method asynchronously. No modifications are needed on the remotable object to achieve this behavior. For more information, see the section "Asynchronous Remoting" in this chapter.
15. **B, C.** Only non-static public methods, properties, and fields participate in remoting. For more information, see the section "Remote Object Activation" in this chapter.

**Suggested Readings and Resources**

1. Visual Studio .NET Combined Help Collection
   * .NET Remoting Overview
   * Remoting Examples
2. Ingo Rammer. *Advanced .NET Remoting*. Apress, 2002.
3. <http://www.iana.org/assignments/port-numbers>—List of assigned port numbers.