8

Working with COM+ Applications in C# 2010

In Depth

COM+ is an evolution of Microsoft Component Object Model (COM) and Microsoft Transaction Server (MTS); where COM represents a technology that enables different software components to communicate with each other, and MTS is a component-based transaction processing system that allows developers to build, deploy, and administer robust network applications. COM+ can be used to develop enterprise level, distributed applications for Windows. You can also use COM+ to build or extend applications that are created using COM, MTS, and other COM based technologies. COM+ automatically performs various resource management tasks, such as thread allocation and security, which was not possible earlier (you had to create specific programs to perform these tasks earlier).

In .NET, the services provided by the COM+ are referred to as Enterprise services, and help you to build and develop robust server applications. COM+ services include Role-based security, Microsoft's queued components, transactions, loosely coupled events, and object pooling.

In this chapter, you learn about Enterprise Services and its concepts, such as transactions, queued components, and loosely coupled events. You also learn to create a simple COM+ application, and then a simple client application that accesses the COM+ application. Next, you learn to create components without inheriting from the `ServicedComponent` class, and set the properties of COM+ applications through the code-behind file. We also discuss different types of deployments, such as automatic deployment, manual deployment, and ClickOnce deployment. Finally, you also learn how to expose COM+ applications as Windows Communication Foundation (WCF) services, and to export object proxies with the help of the Export Wizard.

Let's start with the Enterprise Services.

Enterprise Services

COM+ services, when used with .NET, are called as Enterprise Services. This service is not a new version or enhanced version of COM. The component-based applications are written using COM technology. COM+ also facilitates interoperability between components.

As we already know, a business application is normally divided into three layers—presentation, business, and data services. The presentation layer provides the user interface through which the user interacts with the application to enter or view data. The business layer contains the logic of the business and the business rules; while the data service layer interacts with the databases. You can use Enterprise Services in case of both business and data service layers.

Let's now learn about the contexts.

Contexts

Context is a base functionality provided by Enterprise Services. A context in .NET is a boundary that contains a collection of objects and provides runtime services to the objects that reside inside the process boundary. A proxy server intercepts a method when an object of one context calls the object of another context. It provides the necessary services to the COM+ runtime that are required by the objects of a .NET application, such as call serialization and automatic transaction management.

Now, let's learn about the transactions.

Transactions

A transaction can be defined as a set of statements that are executed as a single unit. Transactions help in maintaining consistency of data by ensuring that either all the statements within the transaction are executed or none of them is executed. COM+ uses the transactions by binding a set of related operations together in a transaction that either completely succeeds or completely fails. Transactions prevent any loss of data in case of power failure or system failure.

Transactions are especially used in applications that use databases as backend and where the chances of failures are considerably high. It brings an application back to its original state, i.e., to a state it was before making any changes to the application. Therefore, transactions help in maintaining a valid state.

All the transactions posses some basic properties, known as ACID (atomic, consistent, isolated, and durable), that are used with respect to databases. These properties were introduced with the purpose of safe sharing of data and keeping a check on inaccurate data from entering the database. When huge number of transactions take place simultaneously, it becomes difficult to maintain accuracy and consistency of data if these ACID properties are not used. Implementing ACID properties ensures error-free transactions. The ACID properties are discussed as follows:

- **Atomic**—Ensures that if any modification, such as insert, update, or delete, occurs in a database, either all the transactions are executed successfully or none of them is executed. This means the part of the transaction that has been committed successfully must be rolled back, in case any part of the transaction is not committed. The property ensures that if the transaction does not commit successfully, the system returns to its original state.
- **Consistent**—Ensures that any changes made in the database are valid and consistent before and after the update is committed successfully. If a transaction is executed successfully, the system returns to a state where all the changes related to the update are successfully made wherever required. However, if the transaction does not execute successfully, the transaction is rolled back and the system returns to its original valid state; therefore, ensuring complete consistency.
- **Isolated**—Ensures that if multiple transactions (called concurrent transactions) occur simultaneously, the transactions are isolated from each other until all the transactions are executed completely.
- **Durable**—Ensures that in case of any power failure, server crash, or any other kind of failure, the system is restored with the updates of the last successfully committed transaction after reboot. It ensures that the changes made to the system are permanent after updating.

COM+ deals with following two types of transactions:

- Distributed Transactions
- Automatic Transactions

Now, let's learn about these transactions one by one.

Distributed Transactions

Enterprise Services offer distributed transactions, in which a transaction can be distributed across multiple database systems. The databases used can be of different vendors, such as SQL Server or Oracle.

All the Enterprise Services transactions are coordinated by the Distributed Transaction Coordinator (DTC), and as a result, listed within the DTC. The DTC requires databases that contain a two-phase commit protocol, called XA protocol. Resource managers, such as SQL Server, Oracle, and the Message Queue server support this protocol.

The two-phase commit protocol contains two phases—the prepare phase and the commit phase. The first a prepare phase occurs when the data is being updated; wherein the participants in the transaction must complete the operations successfully. If any one of the participants cancels or aborts the transaction, a rollback occurs for all the participants. In the commit phase, all the data is updated in the database only if the prepare phase is completed successfully.

Automatic Transactions

Automatic transaction processing is a service provided by COM+ that enables you to configure a class at design time to participate in a transaction at run time. The class must be directly or indirectly derived from the `System.EnterpriseServices.ServicedComponent` class to use automatic transactions.

Table 8.1 displays the attribute values used with the objects in an automatic transaction model:

Table 8.1: Showing the Attribute Values in an Automatic Transaction Model

Attribute Value	Description
`Disabled`	Disables the automatic transactions over the component object. The code to specify the `Disabled` attribute is as follows: `[Transaction(TransactionOption.Disabled)]`
`Supported`	Implies that a component can exist with or without a transaction. This value is useful if the component does not require a transaction, but can call components that use transactions. It can also be called by components that have created transactions. The `Supported` attribute enables the transaction to cross the component, and the components that are calling or are called can participate in the same transaction. The code to specify the `Supported` attribute is as follows: `[Transaction(TransactionOption.Supported)]`
`NotSupported`	Implies that the component does not participate in a transaction, irrespective of whether the caller has a transaction. The code to specify the `NotSupported` attribute is as follows: `[Transaction(TransactionOption.NotSupported)]`
`Required`	Specifies that the object must have a transaction. If a transaction is already created, the object runs within the scope of the transaction; otherwise, if there is no transaction created, it creates a new transaction. The code to specify the Required attribute is as follows: `[Transaction(TransactionOption.Required)]`
`Requires New`	Always creates a new transaction. The component never participates in the same transaction as a caller. The code to specify the `RequiresNew` attribute is as follows: `[Transaction(TransactionOption.RequiresNew)]`

Let's now learn about object pooling and its advantages.

Object Pooling

Object pooling is a service provided by Enterprise Services that helps manage the active objects in an application. When you create an application which uses a component, an object of the component is created for the application. As long as the application accesses the component, the object also exists. The component is destroyed when the application is closed. Object pooling maintains a list of the active objects of the components in a pool. Whenever a request is made by a client to access a component, object pooling enables the client to easily access the specified component from the pool. Therefore, object pooling enables increased performance and scalability of applications, and manages server resources by reusing the objects created.

Object pooling has the following advantages:

- Reduces the time and overhead to create objects, and also minimizes the task of allocating resources
- Minimizes the cost of accessing the server resources, such as maintaining sockets and database connections
- Makes fast access possible in case of re-activating the objects that are enabled just in time
- Makes efficient utilization of the hardware resources, since pool configuration changes with hardware configuration

Next, let's learn about the queued components.

Queued Components

Queuing in simple terms means communication between the sender and the receiver without the need for a connection. In other words, even if the sender and receiver are not present or are not connected, the request-response process is performed without any problem. The messages in the queue are held until the user is ready to accept them. These messages in the queue are saved so that they can be retrieved later when needed.

Similar to other components, a queued component is a service provided by Enterprise Services. This feature is based on Microsoft Message Queuing Service (MSMQ). In a queued component, the client invokes methods with the help of a recorder instead of sending the messages to the message queue. The recorder then creates the messages that are transferred through a message queue to the server application.

Queued components and message queuing helps the client to run an application on the disconnected architecture. The best example can be a laptop-based application that does not always require a connection to the server. Figure 8.1 explains the queuing of messages:

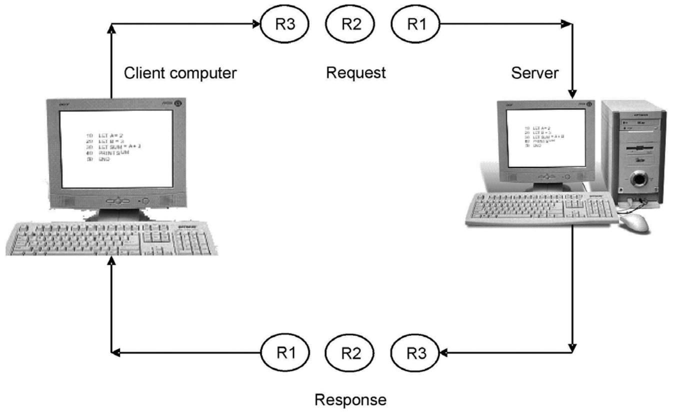


Figure 8.1: Queuing of Messages

Figure 8.2 explains the queued component architecture:

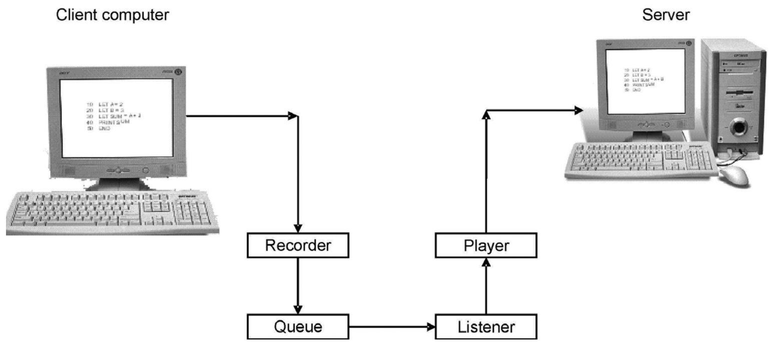


Figure 8.2: Showing the Queued Component Architecture

Let's now learn about the loosely coupled events.

Loosely Coupled Events

- A Loosely Coupled Event (LCE) helps in implementing the COM+ service between the client and the server. The COM+ event services are implemented with the help of four main entities, which are as follows:
- **Event Publisher**—Registers the events that might be needed by subscribers, which request for the event. The information of the event is stored in the COM+ catalog. However, it is not necessary for the event publisher to know the event subscriber. Similarly, subscribers can use the events in the COM+ catalog without knowing the publisher.
- **Event Class**—Acts as a mediator between a publisher and a subscriber, as a direct communication between them is not possible. The Event Class also stores the information about the events in a COM+ catalog, which can later be used by the subscriber.
- **Event Subscriber**—Registers an event that is passed by the LCE service to the subscribers who request for the events.

- **Event Client**—Requests for the events.

The functioning of the loosely coupled event services is shown in Figure 8.3:

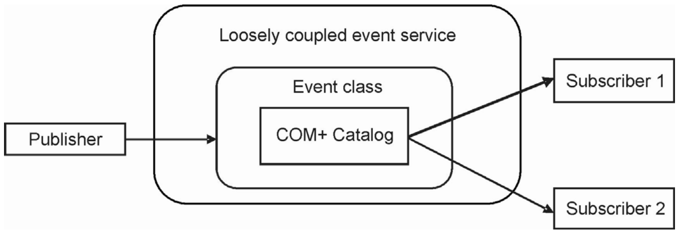


Figure 8.3: Showing the Loosely Coupled Event Service

Next, let's learn about the `ServicedComponent` class and its noteworthy methods.

The ServicedComponent Class

The `ServicedComponent` class is used as a base class for all the classes that need COM+ services. Following is the class hierarchy for the `ServicedComponent` class:

```
System.Object
  System.MarshalByRefObject
        System.ContextBoundObject
                System.EnterpriseServices.ServicedComponent
```

The methods of the `ServicedComponent` class are defined in Table 8.2:

Table 8.2: Showing the Protected Methods of the ServicedComponent Class

Protected Method	Description
`Activate`	Is called when an object is created from a pool. Override this method to customize the initialization code for the objects.
`Deactivate`	Is called when an object is to be deactivated. Override this method to customize the finalization code for the objects.
`CanBePooled`	Is called before the object is placed inside the pool. Override this method to indicate whether or not the object is placed into the pool.
`Construct`	Is called after the constructor is called, which takes a string as a parameter. Override this method to use the constructor string value.

Methods in Table 8.2 are protected and can be overridden in their derived classes.

Application Attributes of Enterprise Services

Enterprise Services provide some `Application` attributes that are needed to build Enterprise Services applications. Table 8.3 lists the `Application` attributes with their description:

Table 8.3: Showing the List of Application Attributes

Protected Attributes	Description
`ApplicationAccessControl`	Turns off the security and allows any user to use a component. Use the following code to set this attribute: `[assembly: ApplicationAccessControl(false)]`
`ApplicationActivation`	Defines whether an application should be used as a library or as a separate process, by using the `ActivationOption.Library` or `ActivationOption.Server` option. Use the following code to set this attribute: `[assembly: ApplicationActivation(ActivationOption.Server)]`
`ApplicationID`	Sets an ID (GUID) for the assembly.

Table 8.3: Showing the List of Application Attributes

Protected Attributes	Description
	Use the following code to set this attribute: `[assembly: ApplicationID("747FC170-1D80-4f45-84CC-42AAB10A6F24")]`
`ApplicationName`	Sets the name of the application when it is displayed in the Component Services Administrative Tool (CSAT). Use the following code to set this attribute: `[assembly: ApplicationName(”Component")]`
`ApplicationQueuing`	Supports queuing for the assembly. Use the following code to set this attribute: `[assembly: ApplicationQueuing]`

Let’s now learn about the ServiceConfig class and its noteworthy properties.

The ServiceConfig Class

The `ServiceConfig` and `ServiceDomain` classes are used for creating services without components. The `ServiceConfig` class configures the context, and the `ServiceDomain` class creates a context.

The class hierarchy for the `ServiceConfig` class is as follows:

```
System.Object
  System.EnterpriseServices.ServiceConfig
```

Table 8.4 displays some of the noteworthy properties of the `ServiceConfig` class:

Table 8.4: Showing the Properties of the ServiceConfig Class

Property	Description
`Inheritance`	Helps you in specifying whether the new context being created should be derived from the existing context or a new context should be created. By default, the new context is derived from the existing context with the `InheritanceOption.Inherit` value. If the value is `InheritanceOption.Ignore`, it means that the existing context is not used.
`Transaction`	Defines the transactional requirements with the help of the `TransactionOption` value.
`TransactionDescription`	Sets a string that describes the transaction.
`TransactionTimeout`	Determines the time until which a transaction lasts before the timeout occurs.

Now, let’s move on to the *Immediate Solutions* section to understand and implement the concepts learned in the *In Depth* section.

Immediate Solutions

Creating a Simple COM+ Application

In this section, we create a simple serviced component application named `Component`. Create a Class Library application in C# named `Component` (available in the CD ROM), since all the COM+ applications must be written as library applications, irrespective of whether they run in the client process or within their own processes. Add a reference of the `System.EnterpriseServices` namespace to the Class Library and also add a declaration `using System.EnterpriseServices` in the `assemblyinfo.cs` and `class1.cs` files. Then, add the `System.Reflection` namespace in the `class1.cs` file to provide an organized view of the loaded types and methods with the capacity to dynamically create and call types.

Next, you need to perform the following broad-level steps to create the Component application:

1. Create a serviced component
2. Create a strong name
3. Adding Assembly to Global Assembly Cache (GAC)
4. Register an assembly

Let's discuss the steps one by one.

Creating a Serviced Component

Listing 8.1 shows how to create a serviced component:

Listing 8.1: Showing the Source Code of the Component Class

```
using System;
using System.Collections.Generic;
using System.Text;
using System.EnterpriseServices;
using System.Reflection;

[assembly: ApplicationName("Component")]
[assembly: ApplicationAccessControl(false)]
[assembly: ApplicationActivation(ActivationOption.Server,SoapVRoot="Component")]
[assembly: Description("Simple Serviced Component Sample")]
[assembly: AssemblyKeyFile("Component.snk")]
namespace Component
{
    public interface IMathOperation
    {
        int Add(int num1, int num2);
    }
    [EventTrackingEnabled(true)]
    [Description("Simple Serviced Component Sample")]

    public class Component : ServicedComponent, IMathOperation
    {
        public Component()
        {

        }

        public int Add(int num1, int num2)
        {
            return num1 + num2;
        }
    }
}
```

In Listing 8.1, the `Class1` class is used to create a serviced component class. In the preceding listing, we have first renamed the `Class1` class as `Component` and then created an interface with the name `IMathOperation`, which contains a method called `Add.` The `Add` method takes two integer parameters in the `Component` class. The `Component` class is derived from the `ServicedComponent` class and also implements the `IMathOperation` interface. The `[EventTrackingEnabled]` attribute monitors the objects from the administrative tool. This attribute should be enabled (true) since, by default, it is false. The `[Description]` attribute shows the text specified for a component in the administrative tool.

NOTE

You need to add a reference of the System.EnterpriseServices namespace to use Enterprise Services.

Let us now create a strong name to share the assembly that needs to be registered using the regasm .NET command utility.

Creating a Strong Name

A strong name can be created to share the assembly by using a tool called `sn`. The following command shows how to provide a strong name to an assembly:

```
    sn –k Component.snk
```

To run the preceding command, open the command prompt of Visual Studio 2010 and navigate to the path location of the application.

NOTE

Change the ComVisible property to true in the assemblyInfo.cs file of the Component application.

After creating a strong name, build the application by selecting the Build→ Build Solution menu option.

Adding Assembly to Global Assembly Cache (GAC)

You can add the assembly to GAC by using the `gacutil` utility. To use the `gacutil` utility open the command prompt of Visual Studio 2010 and navigate to the path location of the assembly. The following command shows how to add an assembly, Component.dll, to GAC:

```
    gacutil /i Component.dll
```

Registering the Assembly

You can register the assembly by using the `regasm` utility. To use the `regasm` utility and execute the following command that shows how to register an assembly, Component.dll:

```
    regasm Component.dll
```

Now, let's create a client application to use the COM+ application.

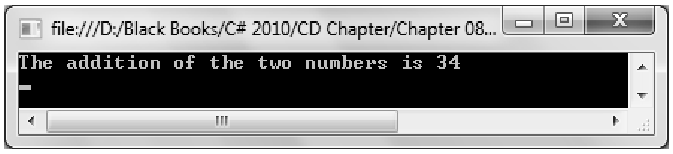
Creating a Simple Client

You need to build a client application to use the serviced component that you have created. This can be built by using a simple Console application called, `Client application` (also available in CD-ROM). Add references of both the `System.EnterpriseServices` namespace and the newly created `Component` namespace to the client project. Create a new object of the `Component` class with the name `com` and invoke the `Add` method with two integer values passed into it. The `Add` method adds the two numbers and returns the result of the addition, which is displayed at a command prompt. The complete code for the Program.cs file of the `Client application` is given in Listing 8.2:

Listing 8.2: Showing the Source Code of the `Program.cs` File

```
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.EnterpriseServices;

    namespace Client_Application
    {
        class Program
        {
            static void Main(string[] args)
            {
                Component.Component com = new Component.Component();
                Console.WriteLine("The addition of the two numbers is " + com.Add(10, 24));
                Console.ReadLine();
            }
        }
    }
```

The output of the preceding code is as shown in Figure 8.4:



```
The addition of the two numbers is 34
_
```

Figure 8.4: Showing the Output after Running the Client Application

Deploying the Assemblies

Assemblies containing serviced components must be deployed. The deployment can be either automatic or manual (through registering the assembly manually). Automatic deployment occurs when the client application, which uses the component, is started; and the COM+ application is automatically configured. This is applicable for all the classes that are derived from the `ServicedComponent` class. The attributes, such as `[Application]`, and the class-level attributes, such as `[EventTrackingEnabled]`, define the configuration characteristics.

In case of automatic deployment, the client application should be a .NET application and should have administrative rights. If the client application is other than a .NET application, its runtime would not include administrative rights. In such a case, automatic deployment is possible only during development time. Automatic deployment is quite advantageous, since, during the development phase, a manual deployment after every single build operation is not required.

Manual deployment refers to the deployment of the .NET application manually using the `regsvcs.exe` utility. The command to deploy a .NET application from the Visual Studio Command Prompt (2010) is as follows:

```
    regsvcs Component.dll
```

The preceding command registers an assembly, named `Component`, as a COM+ application. It also configures the components included within the application according to the attribute values specified. It then creates a type library, which can be used by any client application that accesses the component.

Now that you have deployed the assembly, you can start the component services by opening the CSAT tool, which we discuss next.

Deploying Components Using CSAT

Another method of deploying the COM+ applications is by using the CSAT tool. The CSAT tool is used to enable the deployment of COM+ applications across multiple servers. It is also used to create installation packages for COM+ applications and application proxies. Let's learn how to use the CSAT tool for deploying an application. To do so, you need to add a COM+ application and add an assembly to it. Perform the following steps to add a COM+ application, named Component, in the CSAT tool:

1. Run the CSAT tool by typing `comexp.msc` in the Run dialog box and clicking the OK button, as shown in Figure 8.5:

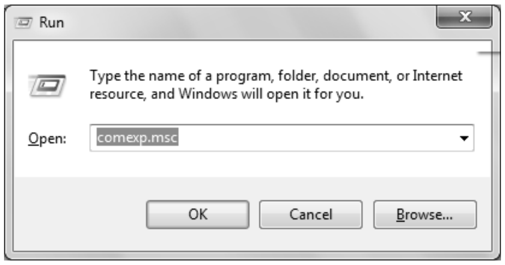

Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open: comexp.msc
OK
Cancel
Browse...

Figure 8.5: Running the CSAT Tool

2. Right click the COM+ Applications node under My Computer and select the New→Application option from the context menu, as shown in Figure 8.6:

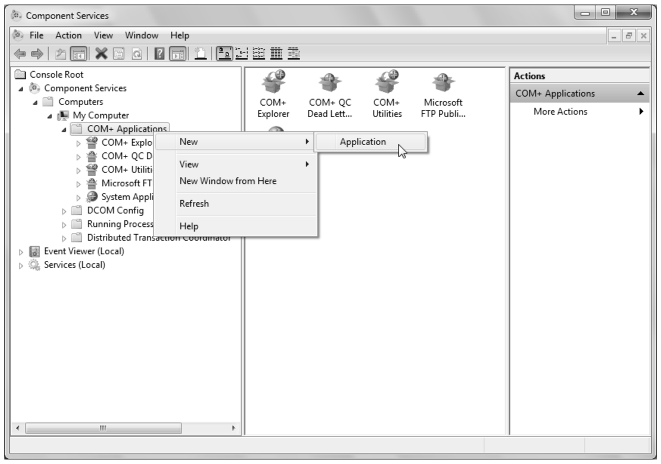


Figure 8.6: Creating New Application in the CSAT Tool

The Welcome to the COM+ Application Install Wizard appears (Figure 8. 7).

3. Click the Next button to continue to install or create the new application, as shown in Figure 8.7:

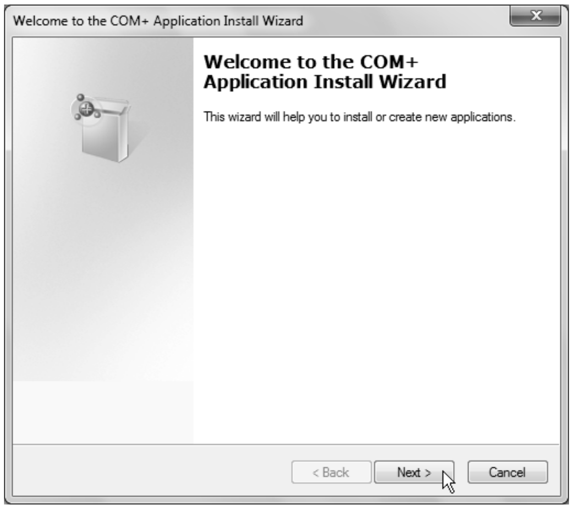

Welcome to the COM+ Application Install Wizard
Welcome to the COM+ Application Install Wizard
This wizard will help you to install or create new applications.
< Back
Next >
Cancel

Figure 8.7: Creating a New Application in the CSAT Tool

The Install or Create a New Application page appears (Figure 8.8).

4. Click the Create an empty application button to continue, as shown in Figure 8.8:

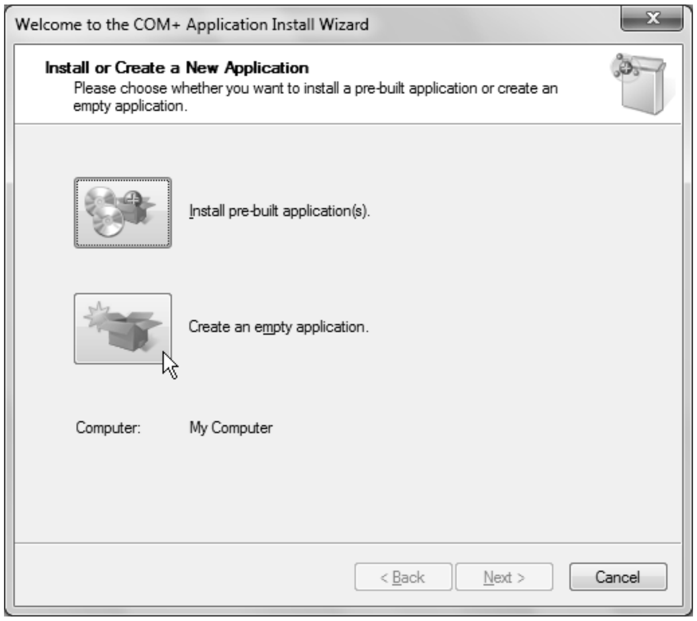
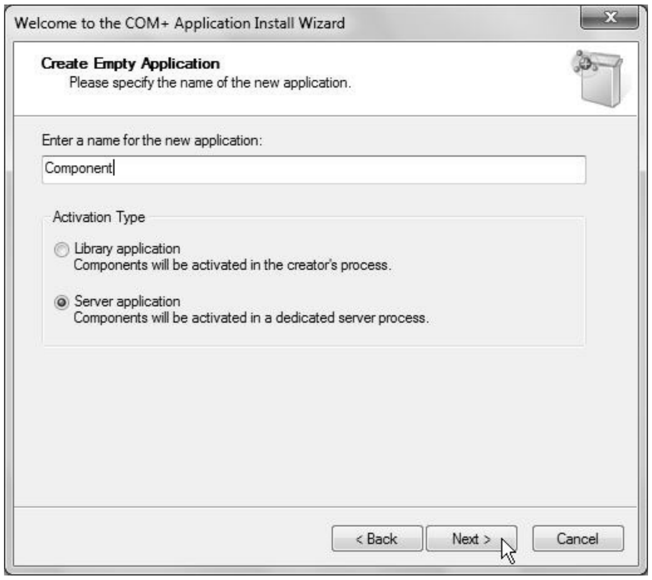

Welcome to the COM+ Application Install Wizard
Install or Create a New Application
Please choose whether you want to install a pre-built application or create an empty application.
Install pre-built application(s).
Create an empty application.
Computer:
My Computer
< Back
Next >
Cancel

Figure 8.8: Clicking the Create an Empty Application Button

The Create Empty Application page appears (Figure 8.9).

5. Type the name of the application in the Enter a name for the new application text box, and select its activation type. In our case, component is the application name and Server application is the Activation Type. Click the Next button to set the application identity, as shown in Figure 8.9:


Welcome to the COM+ Application Install Wizard
Create Empty Application
Please specify the name of the new application.

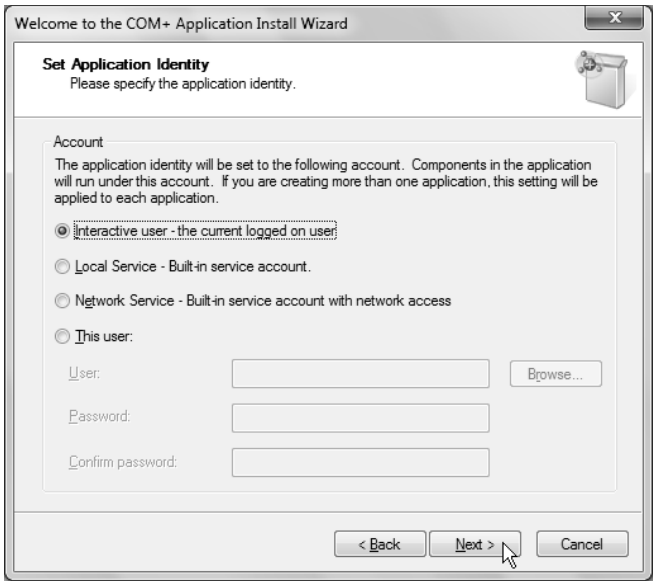
Enter a name for the new application: Component

Activation Type
( ) Library application
Components will be activated in the creator's process.
[✓ Server application]
Components will be activated in a dedicated server process.

Figure 8.9: Entering Application Name and Selecting Activation Type

The Set Application Identity page appears (Figure 8.10).

6. Select the application identity (under which account your application will run). In this case, we select the `Interactive user – the current logged on user` and click the Next button to add application roles, as shown in Figure 8.10:



Welcome to the COM+ Application Install Wizard

Set Application Identity
Please specify the application identity.

Account
The application identity will be set to the following account. Components in the application will run under this account. If you are creating more than one application, this setting will be applied to each application.

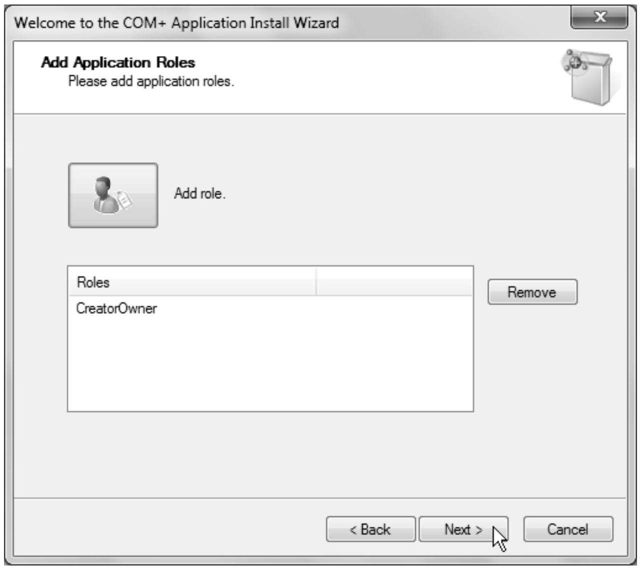
- [✓ Interactive user - the current logged on user]
- Local Service - Built-in service account.
- Network Service - Built-in service account with network access
- This user:

User:
Password:
Confirm password:

Figure 8.10: Selecting Application Identity

The Add Application Roles page appears (Figure 8.11).

7. Add roles for your applications. In this case, use the default settings and click the Next button to add the users to roles, as shown in Figure 8.11:



Welcome to the COM+ Application Install Wizard

Add Application Roles
Please add application roles.

Add role.

Roles
CreatorOwner

Figure 8.11: Adding Roles for the Application

8. Define users for the roles you have added in the previous step. In this case, accept the default settings and click the Next button to finish the wizard, as shown in Figure 8.12:

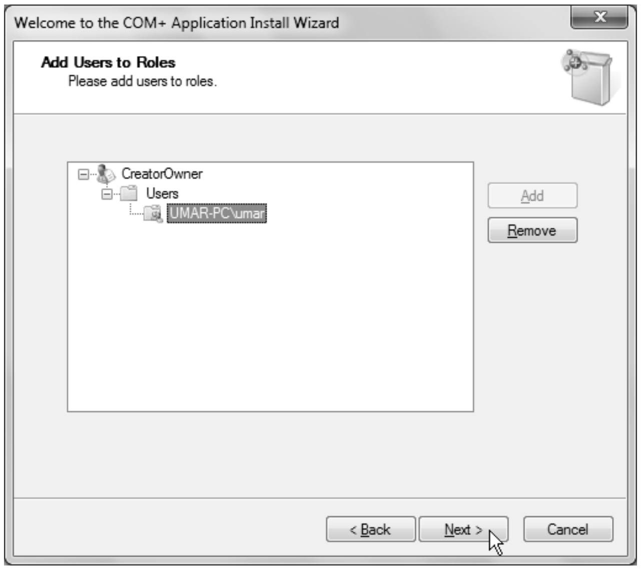


Figure 8.12: Adding Users to the Roles

9. Click the Finish button to close the wizard. This adds an empty application, component, in the CSAT tool, as shown in Figure 8.13:

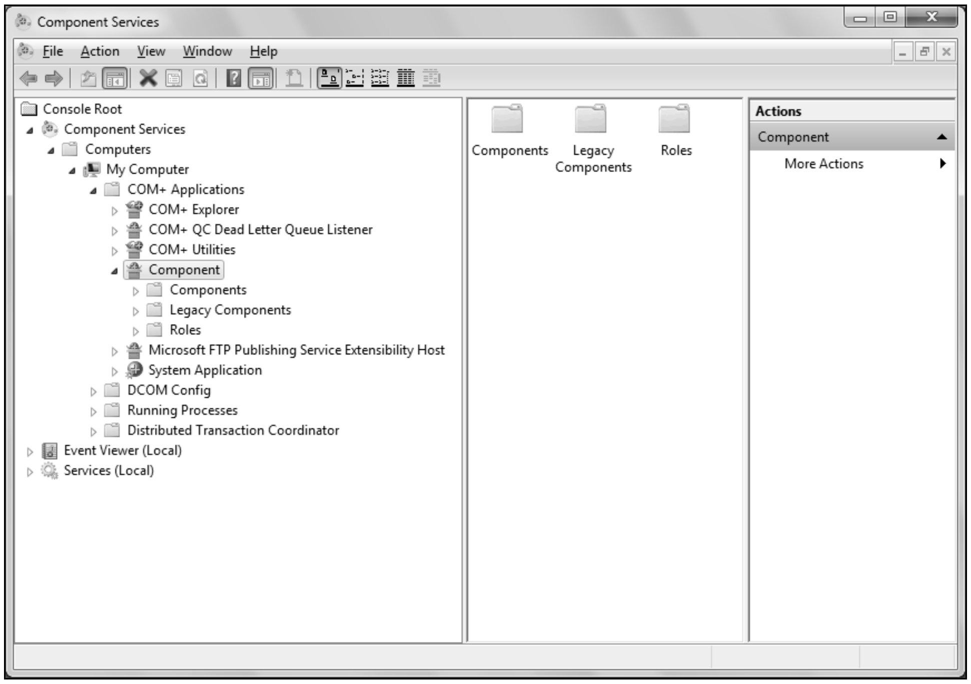


Figure 8.13: Showing the Component Application in the CSAT Tool

After adding an empty COM+ application in CSAT tool, let's perform the following steps to add the `Component.dll` assembly to it:

1. Add the component to the Component application, which we have created earlier, by right-clicking the Components sub-node and select the New→ Component option, as shown in Figure 8.14:

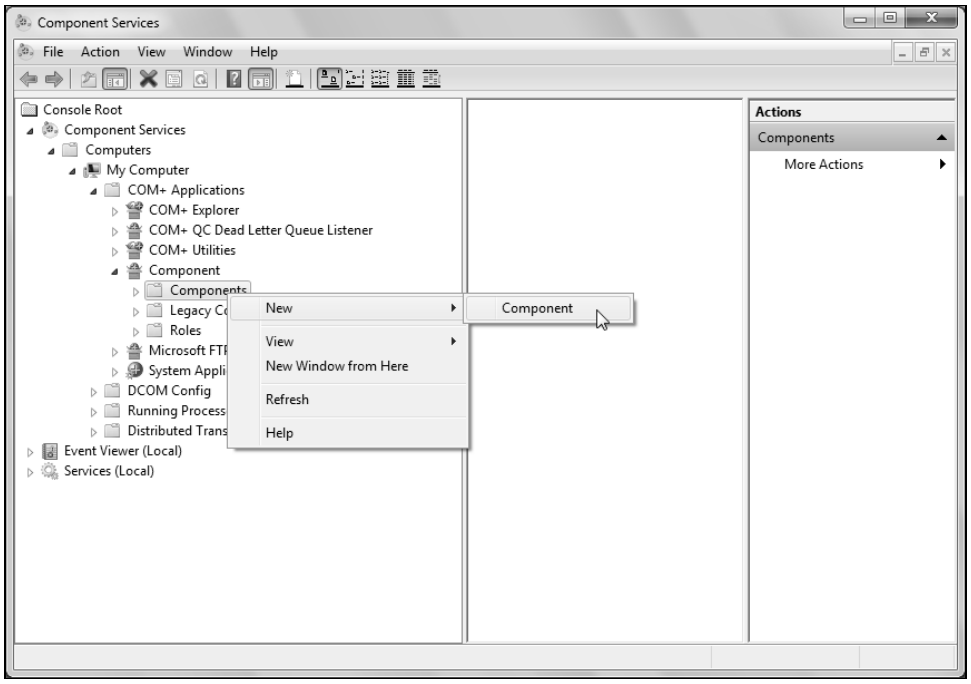
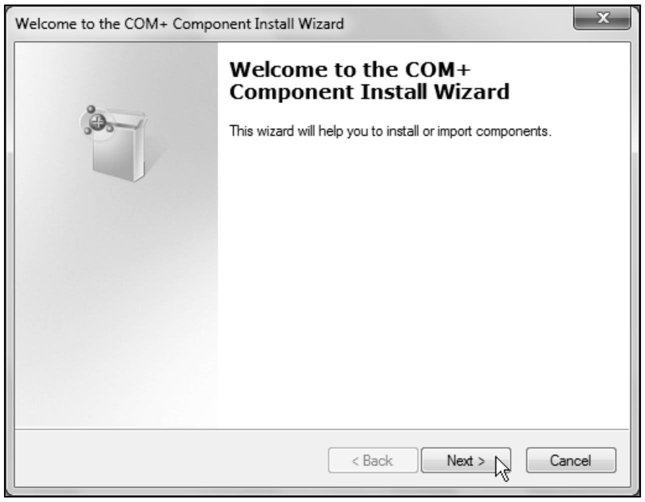


Figure 8.14: Installing a Component in Application

The Welcome to the COM+ Component Install Wizard appears, as shown in Figure 8.15:


Welcome to the COM+ Component Install Wizard

Welcome to the COM+ Component Install Wizard

This wizard will help you to install or import components.

Figure 8.15: Showing the COM+ Component Install Wizard

2. Click the Next button (Figure 8.15) to import or install a component, as shown in Figure 8.16:

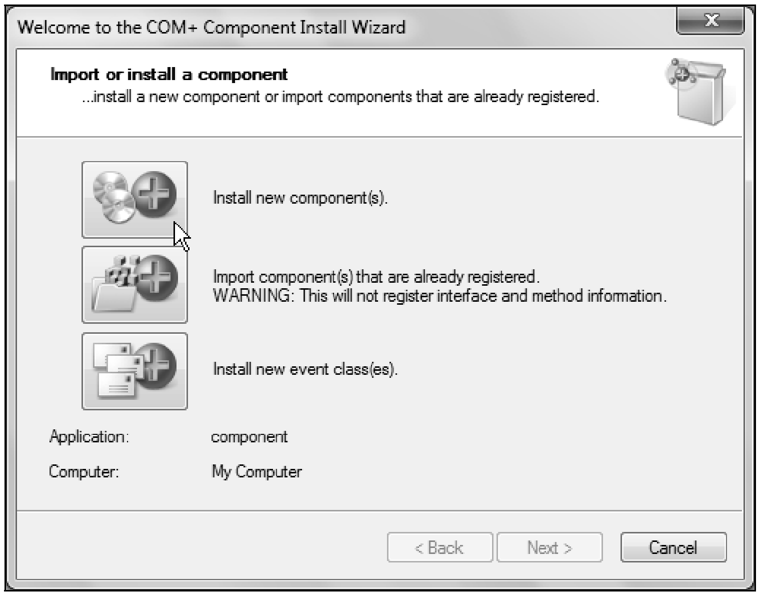

Welcome to the COM+ Component Install Wizard
Import or install a component
...install a new component or import components that are already registered.
Install new component(s).
Import component(s) that are already registered.
WARNING: This will not register interface and method information.
Install new event class(es).
Application: component
Computer: My Computer
< Back
Next >
Cancel

Figure 8.16: Clicking the Install New Component Button

3. Click the Install new component(s) button (Figure 8.16) to open the Select files to install dialog box, as shown in Figure 8.17:

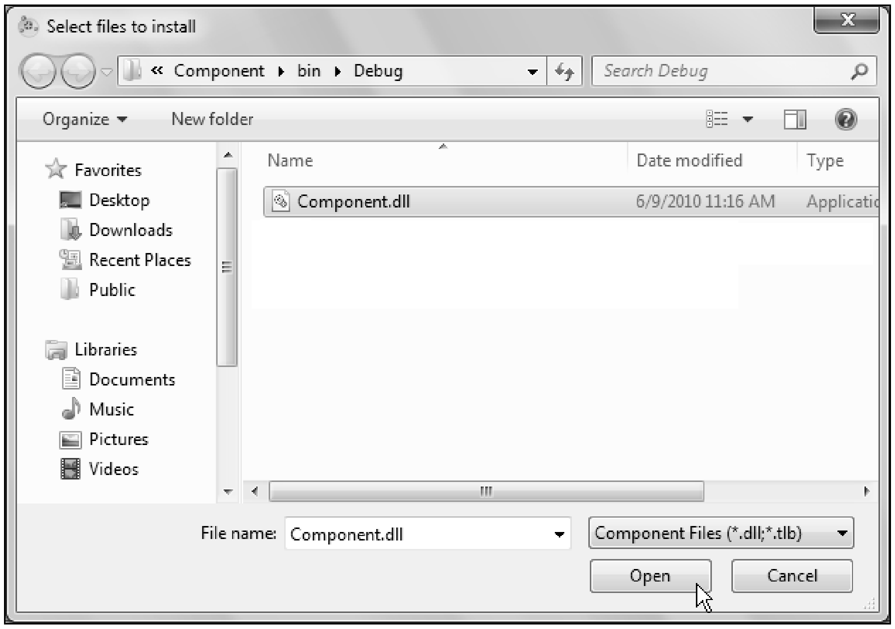
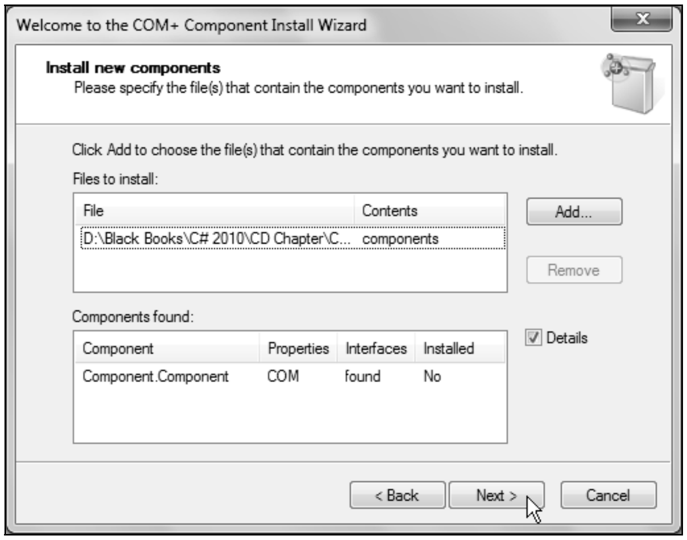


Figure 8.17: Selecting the DLL File

4. Select the .dll file, which is created by your application, in the Select files to install dialog box and click the Open button (Figure 8.18). This opens the Welcome to the COM+ Component Install Wizard, as shown in Figure 8.18:


Welcome to the COM+ Component Install Wizard
Install new components
Please specify the file(s) that contain the components you want to install.
Click Add to choose the file(s) that contain the components you want to install.
Files to install:
Add...
Remove
Components found:
Details
< Back
Next >
Cancel

File	Contents
D:\Black Books\C# 2010\CD Chapter\C...	components

Component	Properties	Interfaces	Installed
Component.Component	COM	found	No

Figure 8.18: Listing the Component to be Installed

5. Click the Next button to continue (Figure 8.18).
6. Click the Finish button in the final page of the wizard. This adds the component to the application with all its interfaces and subscriptions, as shown in Figure 8.19:

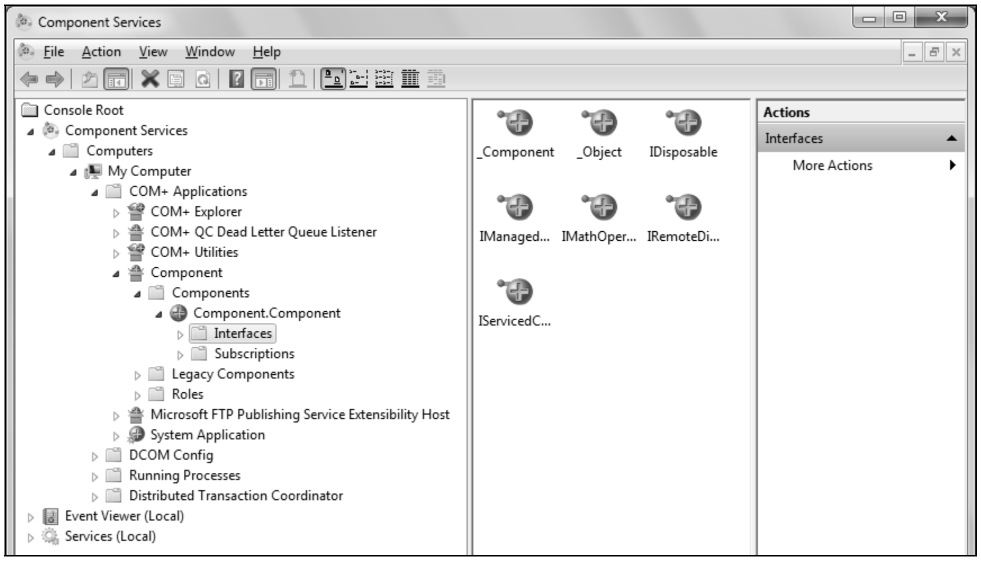


Figure 8.19: Showing the Installed Components

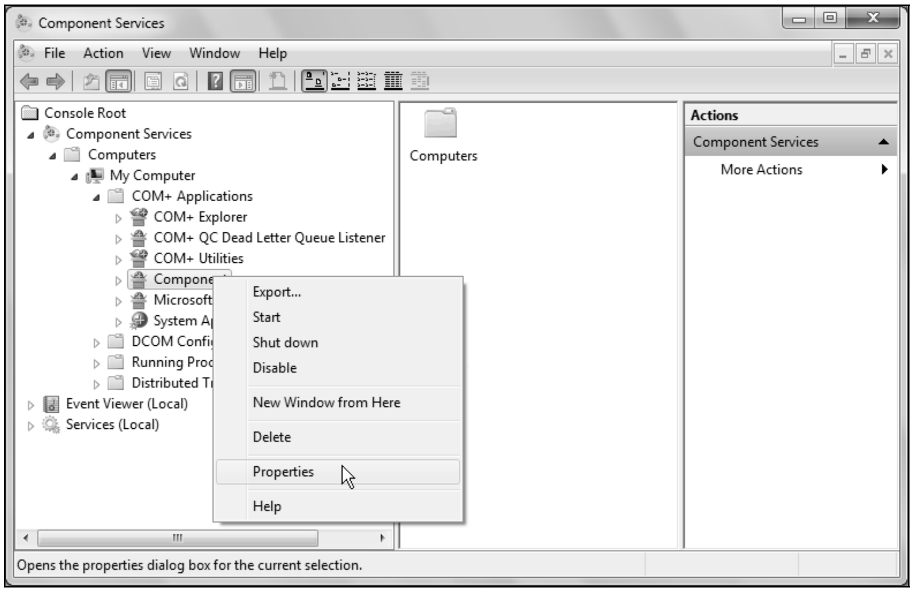
Now, let’s learn how to configure and export a COM+ application using the CSAT tool.

Configuring and Exporting a COM+ Application Using the CSAT Tool

The CSAT tool also allows you to configure the properties of a COM+ application and export it as a service installer. Let’s perform the following steps to configure the properties of the Component application in the CSAT tool:

1. Open the CSAT tool by using the `comexp.msc` command. The Component Services window appears.

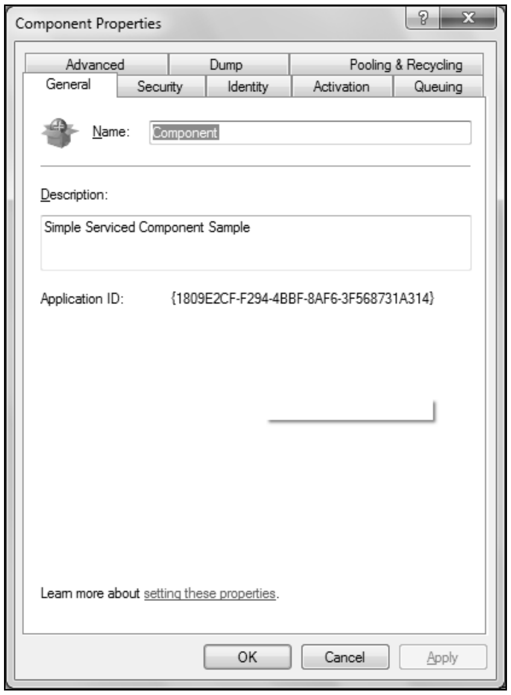
2. Right-click the Component node and select the Properties option to view the properties of any component, as shown in Figure 8.20:



Opens the properties dialog box for the current selection.

Figure 8.20: Selecting the Properties Menu Item in the CSAT Tool

The Component Properties dialog box appears, displaying its General tab, as shown in Figure 8.21:



Name: Component

Description:

Simple Serviced Component Sample

Application ID: {1809E2CF-F294-4BBF-8AF6-3F568731A314}

Learn more about setting these properties.

Figure 8.21: Showing the General Tab of the Component Properties Dialog Box

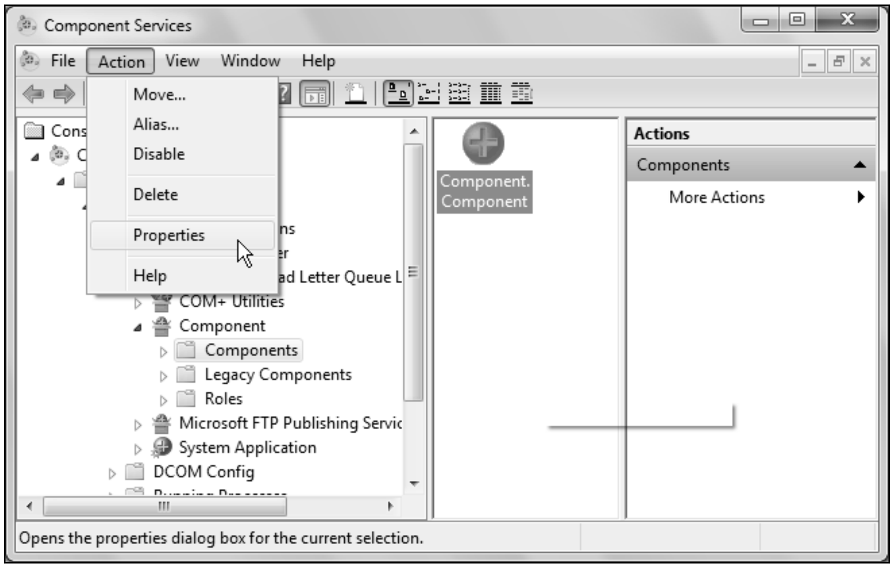
In the Activation tab, you can find that the application is set as a Server application because of the attribute set in the application as `[assembly: ApplicationActivation(ActivationOption.Server)]`. In the Security tab, the Enforce access checks for this application check box is not checked because of the attribute `[assembly: ApplicationAccessControl(false)]`.

Following are some more options that can be set for this application:

- **Security**—Facilitates in enabling or disabling access to users. In case, you enable the security, the access level is set to the application level, the component level, the interface level, or the method level. The messages sent through networks can also be encrypted. However, this affects the performance of the application.
- **Identity**—Configures the Identity tab for the user account that uses the process that hosts the application, when the application is a server application. By default, the user selected is always the interactive user. When you are debugging the application, this setting benefits you but if the application is running on a server, it might not be very useful since it might happen that nobody has logged on. In such a case, the configuration can be changed to a particular user.
- **Activation**—Sets the application as either a library or a server application. The application can run as a Windows service or you can use Simple Object Access Protocol (SOAP) to access the application.
- **Queuing**—Helps the components that use message queuing services.
- **Advanced**—Shuts down an application after a specific period of time after the client becomes inactive. You can also protect your application from any unwanted changes or deletion by locking certain configuration settings.
- **Dump**—Facilitates the user to specify a path in the directory where the dumps can be stored when any application crashes. This is helpful especially for the components developed in C++.
- **Pooling and Recycling**—Configures to restart the application on the basis of the lifetime, memory, and number of calls of an application.

With the help of these options, you can configure the application settings.

3. Select the component in the application and click the Action → Properties menu option to set the configuration properties for the component, as shown in Figure 8.22:



Opens the properties dialog box for the current selection.

Figure 8.22: Selecting the Properties Option for the Component

In the Properties dialog box of the Component.Component class, you can configure the following options:

- **Transactions**—Helps to specify whether or not the component requires transaction.

- **Security**—Checks whether or not the security feature is enabled. If yes, then through this feature, roles for the component can also be specified.
- **Activation**— Sets the object pooling through the Activation tab and it assigns a construction string.
- **Concurrency**— Sets the concurrency to either Required or Requires New attributes, if the component is not thread-safe. At runtime, only one thread is allowed at a time to access the component.

Now, let's learn how to export the component as the MSI installer file using the CSAT tool, which automates the process of component installation.

Perform the following steps to create a MSI installer of the Component application:

1. Right-click the application and select the Export option from the menu to export the component as the MSI installer file using the CSAT tool, as shown in Figure 8.23:

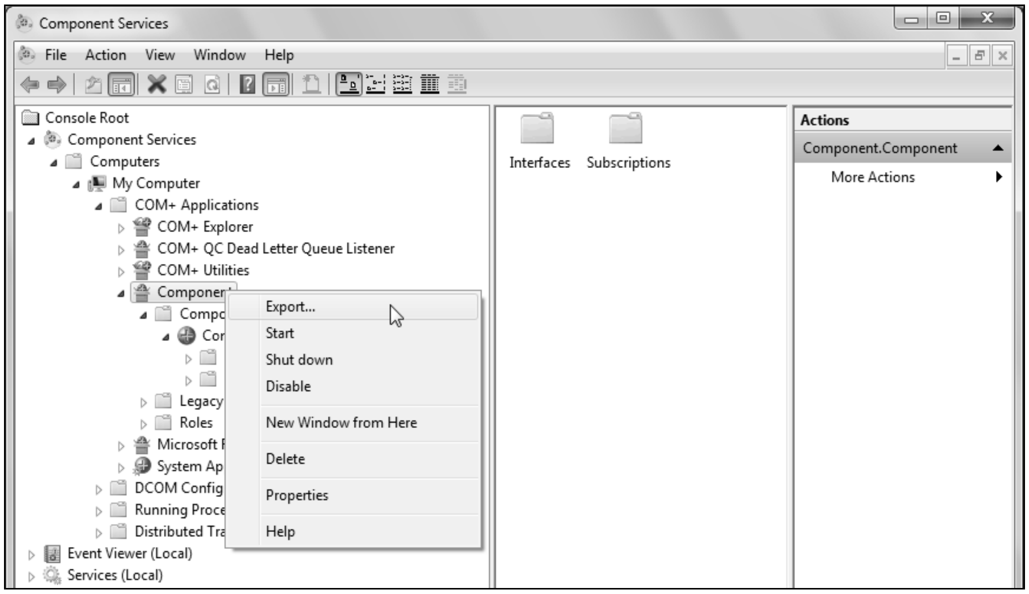
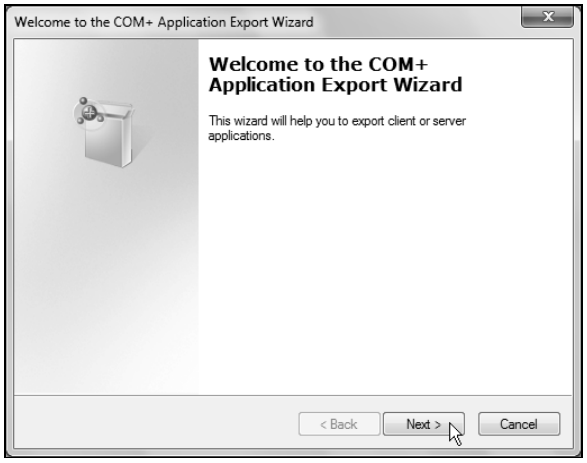


Figure 8.23: Selecting the Export Option

The Welcome to the COM+ Application Export Wizard appears, as shown in Figure 8.24:

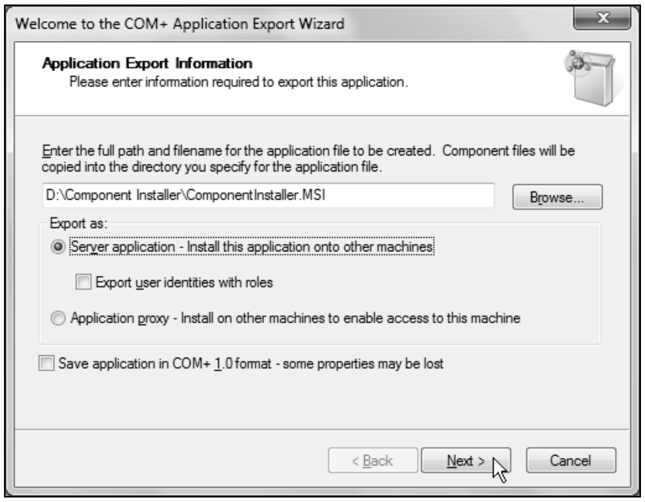


Welcome to the COM+ Application Export Wizard

This wizard will help you to export client or server applications.

Figure 8.24: Showing the COM+ Application Export Wizard

2. Click the Next button to specify the application export information, as shown in Figure 8.25:



Welcome to the COM+ Application Export Wizard

Application Export Information
Please enter information required to export this application.

Enter the full path and filename for the application file to be created. Component files will be copied into the directory you specify for the application file.

D:\Component Installer\ComponentInstaller.MSI

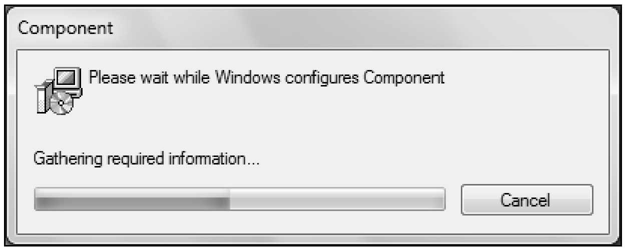
Export as:
- (●) Server application - Install this application onto other machines
  - [ ] Export user identities with roles
- ( ) Application proxy - Install on other machines to enable access to this machine

[ ] Save application in COM+ 1.0 format - some properties may be lost

Figure 8.25: Specifying File Name and Application Type

3. Type the full path and name you want to give to your MSI file. You also need to specify the type of an application. In this case, Enter `D:\Component Installer\ComponentInstaller.MSI` as the full path and select Server application.
4. Click the Next button to generate the MSI installer (Figure 8.25).
5. Click the Finish button to close the wizard.

Let's now install the Component application by using the MSI file on a different computer. To do so, copy the `ComponentInstaller.MSI.cab` and `ComponentInstaller.MSI` files on a computer and double-click the `ComponentInstaller.MSI` file to launch the installer, as shown in Figure 8.26:



Component

Please wait while Windows configures Component

Gathering required information...

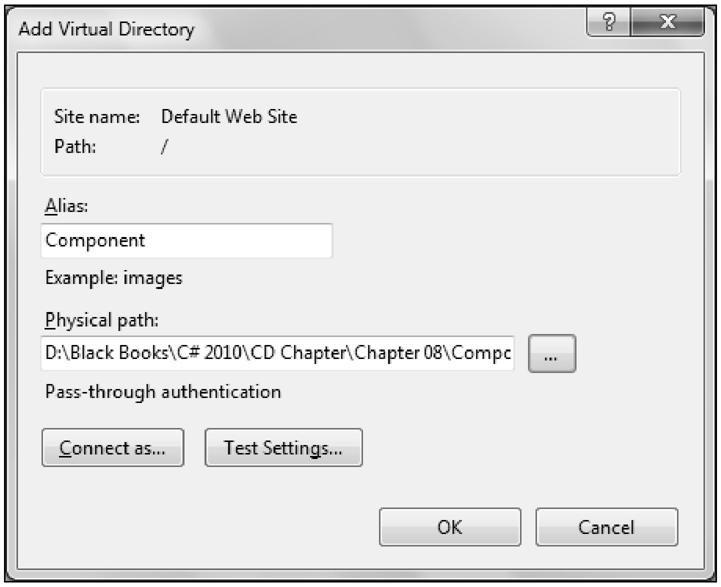
Figure 8.26: Installing Component

The COM+ application is now installed in your computer.

Exposing COM+ Applications as WCF Service

You have learned to create and installed the Component application as a COM+ application. You can use this Component application with the COM or COM+ techniques, only on a local connection. Therefore, to make the Component application available to the other users with Internet connection, .NET 4.0 Framework provides the facility to expose COM+ application as a WCF service. Let's perform the following steps to expose a COM component as a WCF service:

1. Create a folder named `Component` in your computer and then create a new virtual directory in the Internet Information Services (IIS) and browse to the Component folder in the directory you have just created, as shown in Figure 8.27:


Add Virtual Directory

Site name: Default Web Site
Path: /

Alias: Component
Example: images

Physical path: D:\Black Books\C# 2010\CD Chapter\Chapter 08\Compo

Pass-through authentication

Figure 8.27: Adding a Virtual Directory in IIS

2. Open the Service Configuration Editor from Start→ All Programs→ Microsoft Visual Studio 2010→Microsoft Windows SDK Tools→ Service Configuration Editor.
3. Integrate a new COM+ application as a WCF service by selecting the File→Integrate→ COM+ Application option, as shown in Figure 8.28:

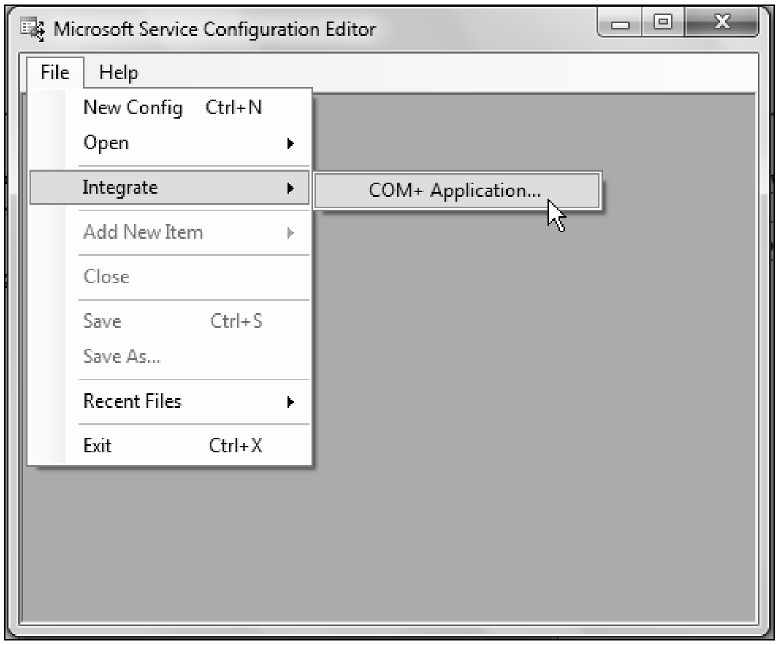
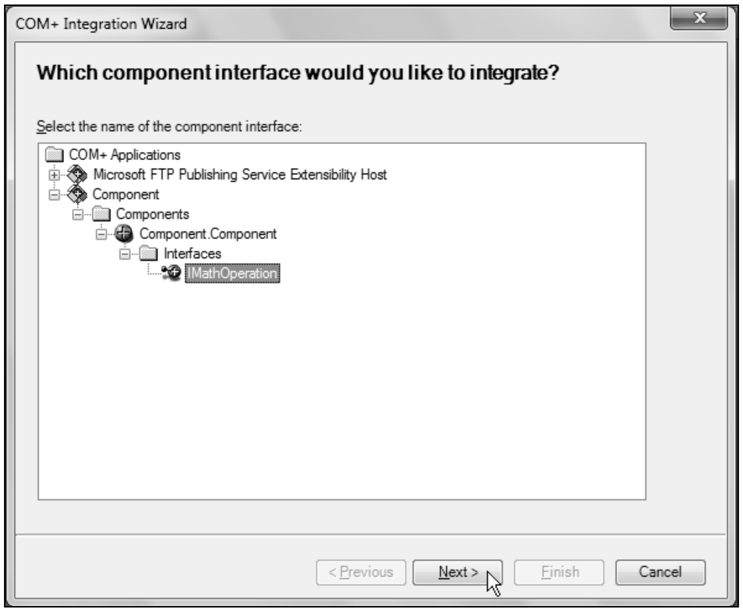


Figure 8.28: Starting COM+ Integration Wizard

The COM+ Integration Wizard appears, as shown in Figure 8.29:


COM+ Integration Wizard

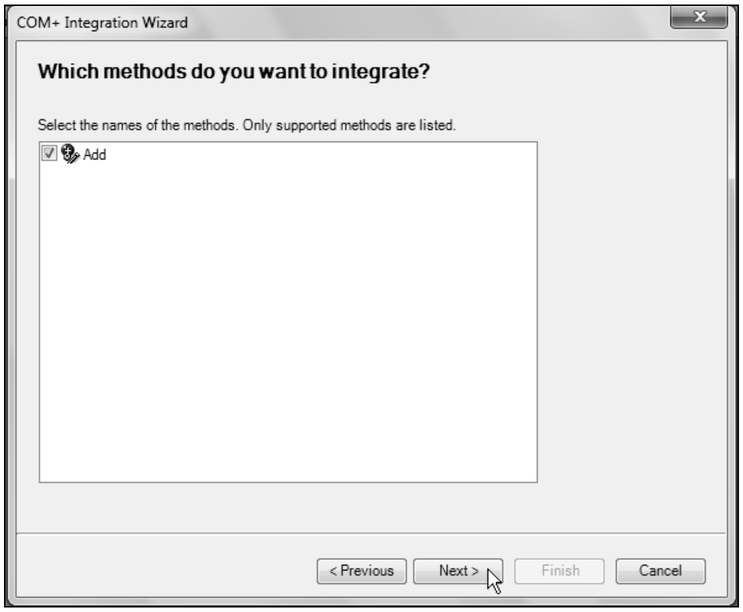
Which component interface would you like to integrate?

Select the name of the component interface:

- COM+ Applications
  - Microsoft FTP Publishing Service Extensibility Host
  - Component
    - Components
      - Component.Component
        - Interfaces
          - IMathOperation

Figure 8.29: Selecting the Interface

4. Select the interface of the component that you are going to publish. In this case, we select IMathOperation (Figure 8.29).
5. Click the Next button (Figure 8.29) to select the method you want to integrate from the next page.
6. Select the Add method that needs to be integrated, as shown in Figure 8.30:


COM+ Integration Wizard

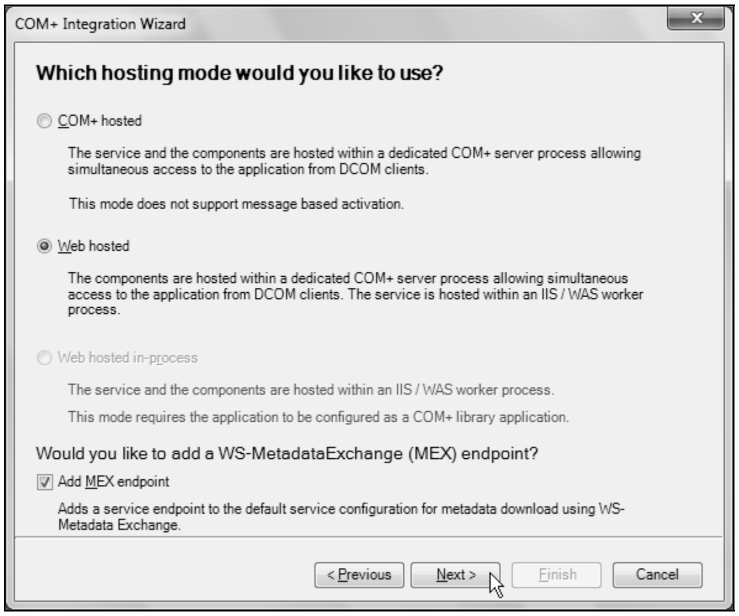
Which methods do you want to integrate?

Select the names of the methods. Only supported methods are listed.

[✓] Add

Figure 8.30: Selecting the Method

7. Click the Next button (Figure 8.30) to select the hosting mode from the next page.
8. Select the Web hosted radio button, as shown in Figure 8.31:


COM+ Integration Wizard

Which hosting mode would you like to use?

○ COM+ hosted

The service and the components are hosted within a dedicated COM+ server process allowing simultaneous access to the application from DCOM clients.

This mode does not support message based activation.

◉ Web hosted

The components are hosted within a dedicated COM+ server process allowing simultaneous access to the application from DCOM clients. The service is hosted within an IIS / WAS worker process.

○ Web hosted in-process

The service and the components are hosted within an IIS / WAS worker process.

This mode requires the application to be configured as a COM+ library application.

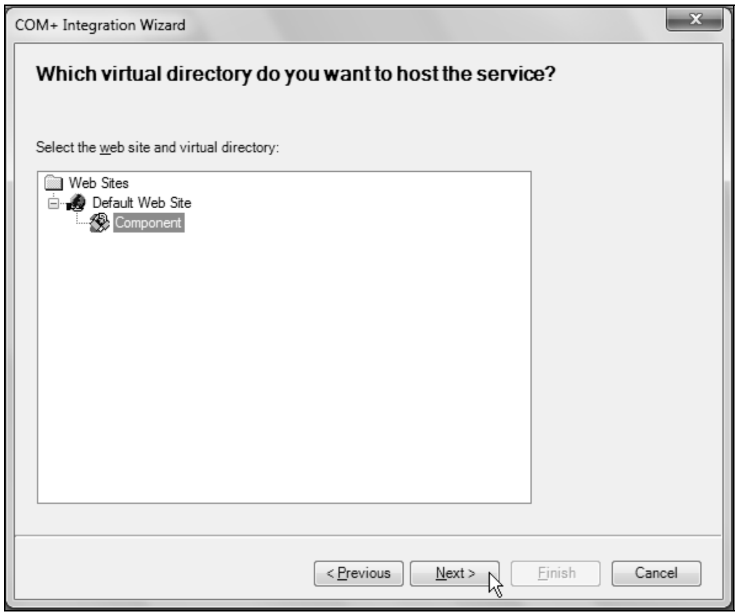
Would you like to add a WS-MetadataExchange (MEX) endpoint?

☑ Add MEX endpoint

Adds a service endpoint to the default service configuration for metadata download using WS-Metadata Exchange.

Figure 8.31: Selecting Hosting Mode

9. Click the Next button (Figure 8.31) to select the virtual directory from the next page.
10. Select the Component virtual directory, which we have created earlier, and click the Next button to continue, as shown in Figure 8.32:

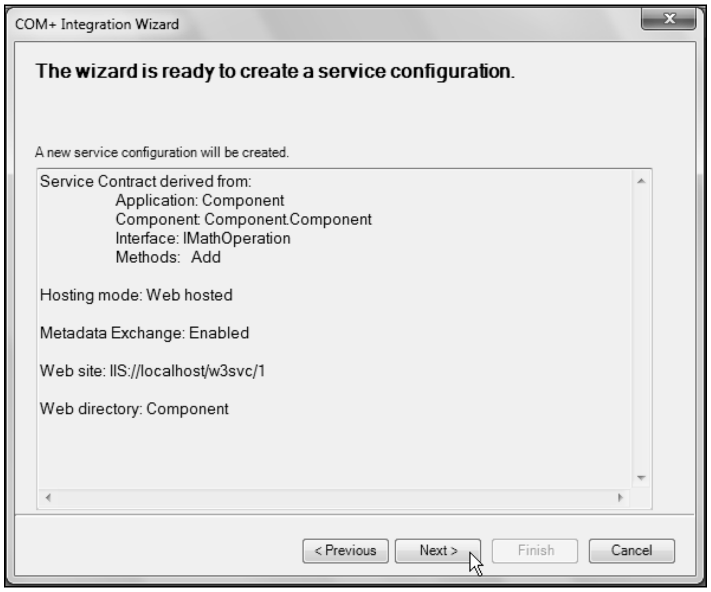

COM+ Integration Wizard

Which virtual directory do you want to host the service?

Select the web site and virtual directory:

Figure 8.32: Selecting Virtual Directory

The next screen displays the information about the service configuration ready to be created, as shown in Figure 8.33:



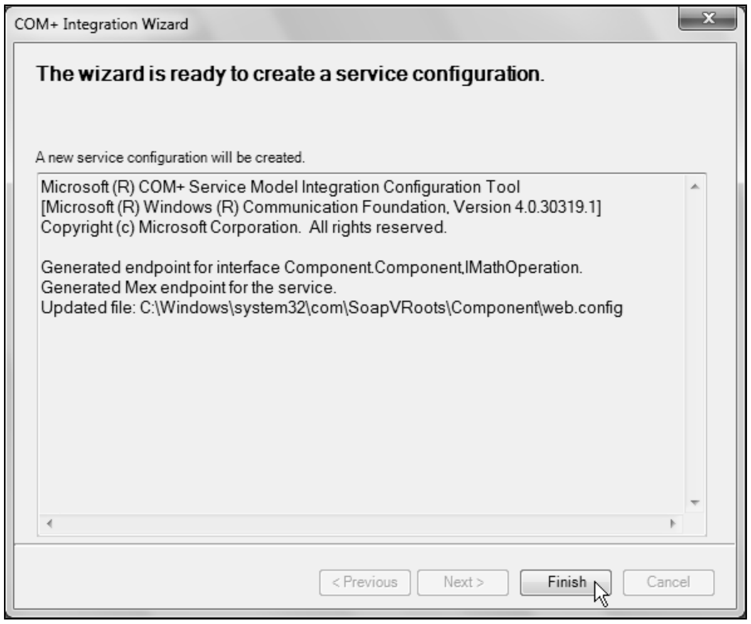
The wizard is ready to create a service configuration.

A new service configuration will be created.

```
Service Contract derived from:
        Application: Component
        Component: Component.Component
        Interface: IMathOperation
        Methods:  Add

Hosting mode: Web hosted

Metadata Exchange: Enabled

Web site: IIS://localhost/w3svc/1

Web directory: Component
```

Figure 8.33: Displaying the Service Configuration Information

11. Click the Next button (Figure 8.33) to continue. The wizard now publishes the WCF service and shows the final screen of the wizard displaying information about the operation performed, as shown in Figure 8.34:



The wizard is ready to create a service configuration.

A new service configuration will be created.

```
Microsoft (R) COM+ Service Model Integration Configuration Tool
[Microsoft (R) Windows (R) Communication Foundation, Version 4.0.30319.1]
Copyright (c) Microsoft Corporation.  All rights reserved.

Generated endpoint for interface Component.Component,IMathOperation.
Generated Mex endpoint for the service.
Updated file: C:\Windows\system32\com\SoapVRoots\Component\web.config
```

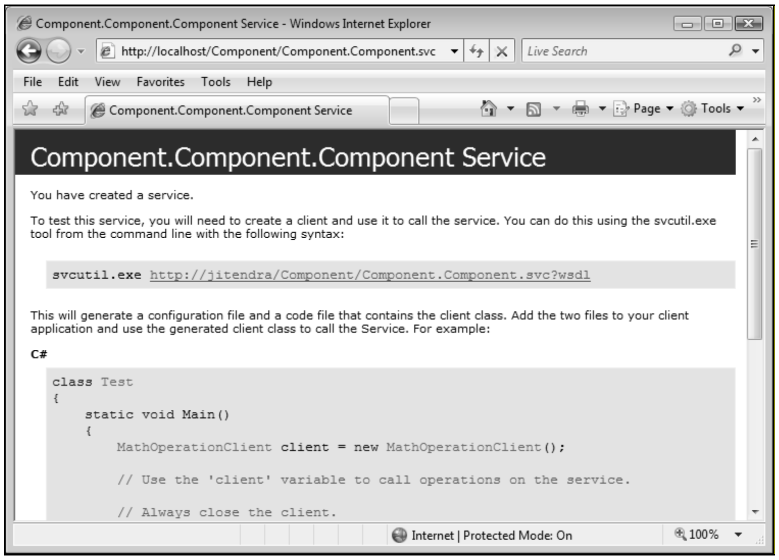
Figure 8.34: Closing the Wizard

12. Click the Finish button (Figure 8.34) to close the wizard.

Now that your service is successfully published, you can check it by typing the following URL in your browser:

```
    http://localhost/Component/Component.Component.svc
```

The output of the preceding URL is shown in Figure 8.35:


Component.Component.Component Service - Windows Internet Explorer
http://localhost/Component/Component.Component.svc
Live Search
File Edit View Favorites Tools Help
Component.Component.Component Service
Page Tools
Internet | Protected Mode: On
100%

Component.Component.Component Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.exe tool from the command line with the following syntax:

```
svcutil.exe http://jitendra/Component/Component.Component.svc?wsdl
```

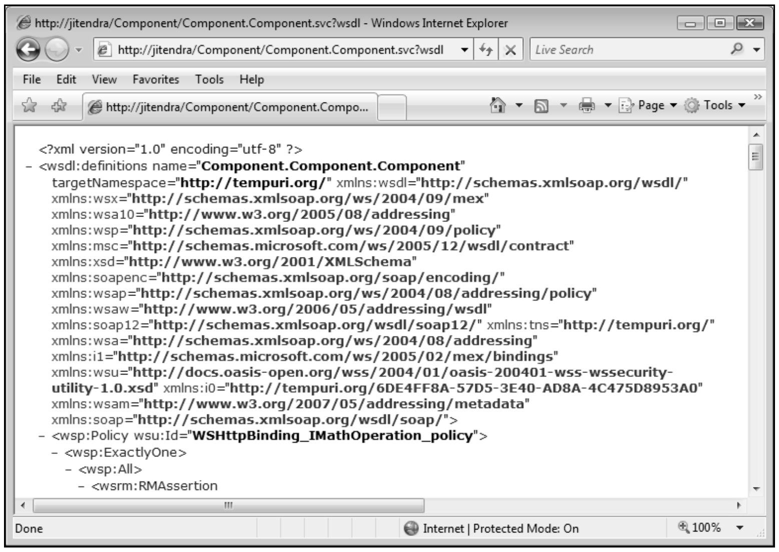
This will generate a configuration file and a code file that contains the client class. Add the two files to your client application and use the generated client class to call the Service. For example:

C#

```
class Test
{
    static void Main()
    {
        MathOperationClient client = new MathOperationClient();

        // Use the 'client' variable to call operations on the service.

        // Always close the client.
```

Figure 8.35: Testing the WCF Service in the Browser

13. Click the link appearing on the page (Figure 8.35). It shows the Web Services Description Language (WSDL) for the WCF service, as shown in Figure 8.36:


http://jitendra/Component/Component.Component.svc?wsdl - Windows Internet Explorer
http://jitendra/Component/Component.Component.svc?wsdl
Live Search
File Edit View Favorites Tools Help
http://jitendra/Component/Component.Compo...
Page Tools
Done
Internet | Protected Mode: On
100%

```
  <?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions name="Component.Component.Component"
    targetNamespace="http://tempuri.org/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex"
    xmlns:wsa10="http://www.w3.org/2005/08/addressing"
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:msc="http://schemas.microsoft.com/ws/2005/12/wsdl/contract"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsap="http://schemas.xmlsoap.org/ws/2004/08/addressing/policy"
    xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:tns="http://tempuri.org/"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:i1="http://schemas.microsoft.com/ws/2005/02/mex/bindings"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
    utility-1.0.xsd" xmlns:i0="http://tempuri.org/6DE4FF8A-57D5-3E40-AD8A-4C475D8953A0"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  - <wsp:Policy wsu:Id="WSHttpBinding_IMathOperation_policy">
    - <wsp:ExactlyOne>
      - <wsp:All>
        - <wsrm:RMAssertion
```

Figure 8.36: Displaying the WSDL for the Web Service

The complete source code of the WSDL file is given in Listing 8.3:

Listing 8.3: Showing the Source Code of the WSDL File

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions name="component.Component.Component"
 targetNamespace="http://tempuri.org/" xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
 xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex"
 xmlns:wsa10=http://www.w3.org/2005/08/addressing
 xmlns:wsp=http://schemas.xmlsoap.org/ws/2004/09/policy
 xmlns:msc=http://schemas.microsoft.com/ws/2005/12/wsdl/contract
```

```
     xmlns:xsd=http://www.w3.org/2001/XMLSchema
     xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
     xmlns:wsap=http://schemas.xmlsoap.org/ws/2004/08/addressing/policy
     xmlns:wsaw=http://www.w3.org/2006/05/addressing/wsdl
     xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:tns=http://tempuri.org/
     xmlns:wsa=http://schemas.xmlsoap.org/ws/2004/08/addressing
     xmlns:i1=http://schemas.microsoft.com/ws/2005/02/mex/bindings
     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
     1.0.xsd" xmlns:i0=http://tempuri.org/6DE4FF8A-57D5-3E40-AD8A-4C475D8953A0
     xmlns:wsam=http://www.w3.org/2007/05/addressing/metadata
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
      <wsp:Policy wsu:Id="WSHttpBinding_IMathOperation_policy">
        <wsp:ExactlyOne>
          <wsp:All>
            <wsrm:RMAssertion xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm/policy">
              <wsrm:InactivityTimeout Milliseconds="600000"/>
              <wsrm:AcknowledgementInterval Milliseconds="200"/>
            </wsrm:RMAssertion>
            <sp:SymmetricBinding
             xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
              <wsp:Policy>
                <sp:ProtectionToken>
                  <wsp:Policy>
                    <sp:SecureConversationToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRe
cipient">
                      <wsp:Policy>
                        <sp:RequireDerivedKeys/>
                        <sp:BootstrapPolicy>
                          <wsp:Policy>
                            <sp:SignedParts>
                              <sp:Body/>
                              <sp:Header Name="To"
                                     Namespace="http://www.w3.org/2005/08/addressing"/>
                              <sp:Header Name="From"
                                     Namespace="http://www.w3.org/2005/08/addressing"/>
                              <sp:Header Name="FaultTo"
                                     Namespace="http://www.w3.org/2005/08/addressing"/>
                              <sp:Header Name="ReplyTo"
                                     Namespace="http://www.w3.org/2005/08/addressing"/>
                              <sp:Header Name="MessageID"
                                     Namespace="http://www.w3.org/2005/08/addressing"/>
                              <sp:Header Name="RelatesTo"
                                     Namespace="http://www.w3.org/2005/08/addressing"/>
                              <sp:Header Name="Action"
                                     Namespace="http://www.w3.org/2005/08/addressing"/>
                            </sp:SignedParts>
                            <sp:EncryptedParts>
                              <sp:Body/>
                            </sp:EncryptedParts>
                            <sp:SymmetricBinding>
                              <wsp:Policy>
                                <sp:ProtectionToken>
                                  <wsp:Policy>
                                    <sp:SpnegoContextToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRe
cipient">
                                      <wsp:Policy>
                                        <sp:RequireDerivedKeys/>
                                      </wsp:Policy>
                                    </sp:SpnegoContextToken>
                                  </wsp:Policy>
                                </sp:ProtectionToken>
                                <sp:AlgorithmSuite>
                                  <wsp:Policy>
                                    <sp:Basic256/>
                                  </wsp:Policy>
                                </sp:AlgorithmSuite>
                                <sp:Layout>
                                  <wsp:Policy>
                                    <sp:Strict/>
                                  </wsp:Policy>
                                </sp:Layout>
                                <sp:IncludeTimestamp/>
                                <sp:EncryptSignature/>
                                <sp:OnlySignEntireHeadersAndBody/>
                              </wsp:Policy>
                            </sp:SymmetricBinding>
                            <sp:Wss11>
                              <wsp:Policy>
                                <sp:MustSupportRefKeyIdentifier/>
                                <sp:MustSupportRefIssuerSerial/>
                                <sp:MustSupportRefThumbprint/>
                                <sp:MustSupportRefEncryptedKey/>
                              </wsp:Policy>
                            </sp:Wss11>
                            <sp:Trust10>
                              <wsp:Policy>
                                <sp:MustSupportIssuedTokens/>
                                <sp:RequireClientEntropy/>
                                <sp:RequireServerEntropy/>
                              </wsp:Policy>
                            </sp:Trust10>
                          </wsp:Policy>
                        </sp:BootstrapPolicy>
                      </wsp:Policy>
                    </sp:SecureConversationToken>
                  </wsp:Policy>
                </sp:ProtectionToken>
                <sp:AlgorithmSuite>
                  <wsp:Policy>
                    <sp:Basic256/>
                  </wsp:Policy>
                </sp:AlgorithmSuite>
                <sp:Layout>
                  <wsp:Policy>
                    <sp:Strict/>
                  </wsp:Policy>
                </sp:Layout>
                <sp:IncludeTimestamp/>
                <sp:EncryptSignature/>
                <sp:OnlySignEntireHeadersAndBody/>
              </wsp:Policy>
            </sp:SymmetricBinding>
            <sp:Wss11 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
              <wsp:Policy>
                <sp:MustSupportRefKeyIdentifier/>
                <sp:MustSupportRefIssuerSerial/>
                <sp:MustSupportRefThumbprint/>
                <sp:MustSupportRefEncryptedKey/>
              </wsp:Policy>
```

```
        </sp:Wss11>
        <sp:Trust10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:MustSupportIssuedTokens/>
            <sp:RequireClientEntropy/>
            <sp:RequireServerEntropy/>
          </wsp:Policy>
        </sp:Trust10>
        <wsaw:UsingAddressing/>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <wsp:Policy wsu:Id="WSHttpBinding_IMathOperation_Add_Input_policy">
    <wsp:ExactlyOne>
      <wsp:All>
        <sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <sp:Body/>
          <sp:Header Name="Sequence"
                Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
          <sp:Header Name="SequenceAcknowledgement"
                Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
          <sp:Header Name="AckRequested"
                Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
          <sp:Header Name="To" Namespace="http://www.w3.org/2005/08/addressing"/>
          <sp:Header Name="From" Namespace="http://www.w3.org/2005/08/addressing"/>
          <sp:Header Name="FaultTo" Namespace="http://www.w3.org/2005/08/addressing"/>
          <sp:Header Name="ReplyTo" Namespace="http://www.w3.org/2005/08/addressing"/>
          <sp:Header Name="MessageID" Namespace="http://www.w3.org/2005/08/addressing"/>
          <sp:Header Name="RelatesTo" Namespace="http://www.w3.org/2005/08/addressing"/>
          <sp:Header Name="Action" Namespace="http://www.w3.org/2005/08/addressing"/>
        </sp:SignedParts>
        <sp:EncryptedParts
                xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <sp:Body/>
        </sp:EncryptedParts>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <wsp:Policy wsu:Id="WSHttpBinding_IMathOperation_Add_output_policy">
    <wsp:ExactlyOne>
      <wsp:All>
        <sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <sp:Body/>
          <sp:Header Name="Sequence"
                Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
          <sp:Header Name="SequenceAcknowledgement"
                Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
          <sp:Header Name="AckRequested"
                Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
          <sp:Header Name="To" Namespace="http://www.w3.org/2005/08/addressing"/>
          <sp:Header Name="From" Namespace="http://www.w3.org/2005/08/addressing"/>
          <sp:Header Name="FaultTo" Namespace="http://www.w3.org/2005/08/addressing"/>
          <sp:Header Name="ReplyTo" Namespace="http://www.w3.org/2005/08/addressing"/>
          <sp:Header Name="MessageID" Namespace="http://www.w3.org/2005/08/addressing"/>
          <sp:Header Name="RelatesTo" Namespace="http://www.w3.org/2005/08/addressing"/>
          <sp:Header Name="Action" Namespace="http://www.w3.org/2005/08/addressing"/>
        </sp:SignedParts>
        <sp:EncryptedParts
                xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <sp:Body/>
```

```
            </sp:EncryptedParts>
          </wsp:All>
        </wsp:ExactlyOne>
      </wsp:Policy>
      <wsdl:import namespace=http://tempuri.org/6DE4FF8A-57D5-3E40-AD8A-4C475D8953A0
     location="http://jitendra/Component/Component.Component.svc?wsdl=wsdl0"/>
      <wsdl:import namespace=http://schemas.microsoft.com/ws/2005/02/mex/bindings
            location="http://jitendra/Component/Component.Component.svc?wsdl=wsdl2"/>
      <wsdl:types/>
      <wsdl:binding name="WSHttpBinding_IMathOperation" type="i0:IMathOperation">
        <wsp:PolicyReference URI="#WSHttpBinding_IMathOperation_policy"/>
        <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="Add">
          <soap12:operation soapAction="http://tempuri.org/6DE4FF8A-57D5-3E40-AD8A-
4C475D8953A0/IMathOperation/Add" style="document"/>
          <wsdl:input>
            <wsp:PolicyReference URI="#WSHttpBinding_IMathOperation_Add_Input_policy"/>
            <soap12:body use="literal"/>
          </wsdl:input>
          <wsdl:output>
            <wsp:PolicyReference URI="#WSHttpBinding_IMathOperation_Add_output_policy"/>
            <soap12:body use="literal"/>
          </wsdl:output>
        </wsdl:operation>
      </wsdl:binding>
      <wsdl:service name="component.Component.Component">
        <wsdl:port name="WSHttpBinding_IMathOperation"
            binding="tns:WSHttpBinding_IMathOperation">
          <soap12:address
            location="http://jitendra/Component/Component.Component.svc/IMathOperation"/>
          <wsa10:EndpointReference>

<wsa10:Address>http://jitendra/Component/Component.Component.svc/IMathOperation</wsa10:Addres
s>
            <Identity xmlns="http://schemas.xmlsoap.org/ws/2006/02/addressingidentity">
              <Spn>host/Jitendra</Spn>
            </Identity>
          </wsa10:EndpointReference>
        </wsdl:port>
        <wsdl:port name="MetadataExchangeHttpBinding_IMetadataExchange"
                    binding="i1:MetadataExchangeHttpBinding_IMetadataExchange">
          <soap12:address location="http://jitendra/Component/Component.Component.svc/mex"/>
          <wsa10:EndpointReference>

<wsa10:Address>http://jitendra/Component/Component.Component.svc/mex</wsa10:Address>
          </wsa10:EndpointReference>
        </wsdl:port>
      </wsdl:service>
    </wsdl:definitions>
```

Creating Services without Components

Let's create a new Windows Forms application, named `ServicesWithoutComponents` (also available on the CD-ROM), and add another Class Library project to the solution, named `TransactionComponent,` to create a service without components. Now, perform the following steps to create the application:

1. Create an SQL Server database, Products, in your computer with two tables, Products and ProductsType. The schema for the Products table is given in Table 8.5:

Table 8.5: Showing the Schema for the Products Table

Field Name	Type	Size	Allow Nulls
Productid	int	-	[No]
Name	varchar	50	[Yes]
Description	varchar	500	[Yes]

NOTE

In addition, you need to set the IsIdentity property of the Productid column of the Products table to Yes.

Table 8.6 displays the schema for the ProductsType table:

Table 8.6: Showing the Schema for the ProductsType Table

Field Name	Type	Size	Allow Nulls	
Productid	int	-	[No]	
Producttype	varchar	250	[Yes]	

2. In the Class Library project, add three class files named `Products.cs`, `ProductType.cs`, and `TransactionClass.cs`. Add the code, given in Listing 8.4, in the `Products.cs` file:

Listing 8.4: Showing the Source Code for the `Products.cs` File

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;

namespace TransactionComponent
{
    public class Products
    {
        public int AddProducts(string name, string description)
        {
            int productID;

            string connString = @"Data Source=umar-pc;Initial Catalog=Products;Integrated
            Security=True";

            string commandName = "AddProducts";

            using (SqlConnection conn = new SqlConnection(connString))
            {
                conn.Open();
                SqlCommand command = new SqlCommand(commandName, conn);
                command.CommandType = CommandType.StoredProcedure;
                SqlParameter paramName = new SqlParameter("@Name", SqlDbType.VarChar,50);
                paramName.Direction = ParameterDirection.Input;
                paramName.Value = name;
                command.Parameters.Add(paramName);
                SqlParameter paramDesc = new
                 SqlParameter("@Description",SqlDbType.VarChar, 250);
                paramDesc.Direction = ParameterDirection.Input;
                paramDesc.Value = description;
                command.Parameters.Add(paramDesc);
                SqlParameter paramReturnValue = new
                 SqlParameter("@@identity",SqlDbType.VarChar, 250);
                paramReturnValue.Direction = ParameterDirection.ReturnValue;
                command.Parameters.Add(paramReturnValue);
                command.ExecuteNonQuery();
                productID = (int)command.Parameters["@@identity"].Value;
            }
            return productID;
        }
    }
}
```

NOTE

Change the connection string according to your system setting.

In the preceding code, the `Products` class contains a method, `AddProducts`, which uses a stored procedure, `AddProducts`, to add a new product in the database. The stored procedure returns an identity value to the calling method for adding new products in the database.

3. Create the `AddProducts` stored procedure in SQL Server 2008, as shown in Listing 8.5:

Listing 8.5: Showing the Code for Creating the AddProducts Stored Procedure

```
CREATE proc AddProducts
@Name varchar(50) ,
@Description varchar(500)
as
insert into Products([Name], Description) Values(@Name, @Description)
return @@identity
```

4. Add the code, shown in Listing 8.6, to create the `ProductType` class in the `ProductType.cs` file:

Listing 8.6: Showing the Source Code of the ProductType.cs File

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;

namespace TransactionComponent
{
    public class ProductType
    {
        public int AddProductType(int productID, string productType)
        {
            int productTypeID;
            string connString = @"Data Source=umar-pc;Initial Catalog=Products;Integrated
                 Security=True";
            string commandName = "AddProductTypes";
            using (SqlConnection conn = new SqlConnection(connString))
            {
                conn.Open();
                SqlCommand command = new SqlCommand(commandName, conn);
                command.CommandType = CommandType.StoredProcedure;
                SqlParameter paramProductID = new
                 SqlParameter("@ProductID",SqlDbType.Int);
                paramProductID.Direction = ParameterDirection.Input;
                paramProductID.Value = productID;
                command.Parameters.Add(paramProductID);
                SqlParameter paramProductType = new
                 SqlParameter("@ProductType",SqlDbType.VarChar, 50);
                paramProductType.Direction = ParameterDirection.Input;
```

```
                    paramProductType.Value = productType;
                    command.Parameters.Add(paramProductType);
                    SqlParameter paramReturnValue = new
                     SqlParameter("@@identity",SqlDbType.VarChar, 250);
                    paramReturnValue.Direction = ParameterDirection.ReturnValue;
                    command.Parameters.Add(paramReturnValue);
                    command.ExecuteNonQuery();
                    productTypeID = (int)command.Parameters["@@identity"].Value;
                }
                return productTypeID;
            }
        }
    }
```

As you can see in the given code, both the classes—`Products` and `ProductType`—are similar to each other. The `ProductType` class also contains one single method, `AddProductType`, which simply adds the product type details in the database. Similar to the `AddProducts` method, the `AddProductType` method uses a stored procedure called `AddProductTypes` and also returns an ID of the newly added product type to the calling method.

5. Create the `AddProductTypes` stored procedure in SQL Server 2008, as shown in Listing 8.7:

Listing 8.7: Showing the Code for Creating the AddProducts Stored Procedure

```
CREATE proc AddProductTypes
@ProductID int ,
@ProductType varchar(250)
as
insert into ProductsType(ProductID, ProductType) Values(@ProductID, @ProductType)
return @@identity
```

6. Add the code for the `TransactionClass` class, shown in Listing 8.8, , which ensures that the details related to the products and product types are added as a single transaction:

Listing 8.8: Showing the Source Code of the `TransactionClass.cs` File

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.EnterpriseServices;

namespace TransactionComponent
{
    public class TransactionClass
    {
        public string AddDetails(string prodName, string prodDescription, string
         prodType)
        {
            ServiceConfig config = new ServiceConfig();
            config.Transaction = TransactionOption.Required;
            ServiceDomain.Enter(config);
            try
            {
                Products prod = new Products();
                int ProdID = prod.AddProducts(prodName, prodDescription);
                ProductType pType = new ProductType();
                pType.AddProductType(ProdID, prodType);
            }
            catch
            {
                throw;
            }
            finally
```

```
                {
                    ServiceDomain.Leave();
                }
                return "Data entered successfully";
            }

        }
    }
```

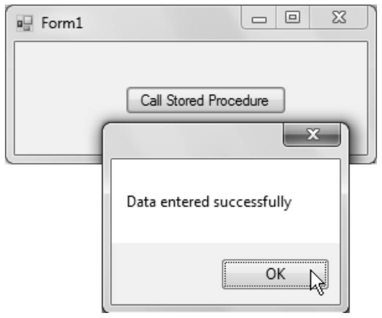
In the preceding code, an object of the `ServiceConfig` class is created, which invokes the `AddProducts` method, when the object is created. After that, the `Transaction` property is set to `TransactionOption.Required`, which ensures that the object runs within the scope of the transaction. Now, to enter into the context, the `ServiceDomain.Enter` method is used. Invoke the `AddProducts` and `AddProductType` methods using their corresponding objects. As a transaction is used to maintain consistency of data, the statements between the `Enter` and `Leave` methods ensure that either the transaction is commited successfully, or if any exception arises during the execution, it is rolled back.

7. Add a reference of the `TransactionComponent` component in the `ServicesWithoutComponents` application to test the application component created.
8. Add a `Button` control to the Form1 form in the `ServicesWithoutComponents` application and change its Text property to Call Stored Procedure.
9. Add the following code snippet on the `Click` event of the Button control:

```
private void button1_Click(object sender, EventArgs e) {
   TransactionClass trans = new TransactionClass();
   MessageBox.Show(trans.AddDetails("Table Lamp", "A table lamp with
   the picture of flowers", "Show piece"));
}
```

As you can see in the code, the `AddDetails` method of the `TransactionClass` class returns a string value if the method executes successfully.

10. Add the TransactionComponent namespace in the TransactionClass class file.
11. Set the ServicesWithoutComponents project as the startup project and Form1 as the startup form.
12. Press the F5 key on the keyboard to execute the application and click the Call Stored Procedure button to see the result, as shown in Figure 8.37:



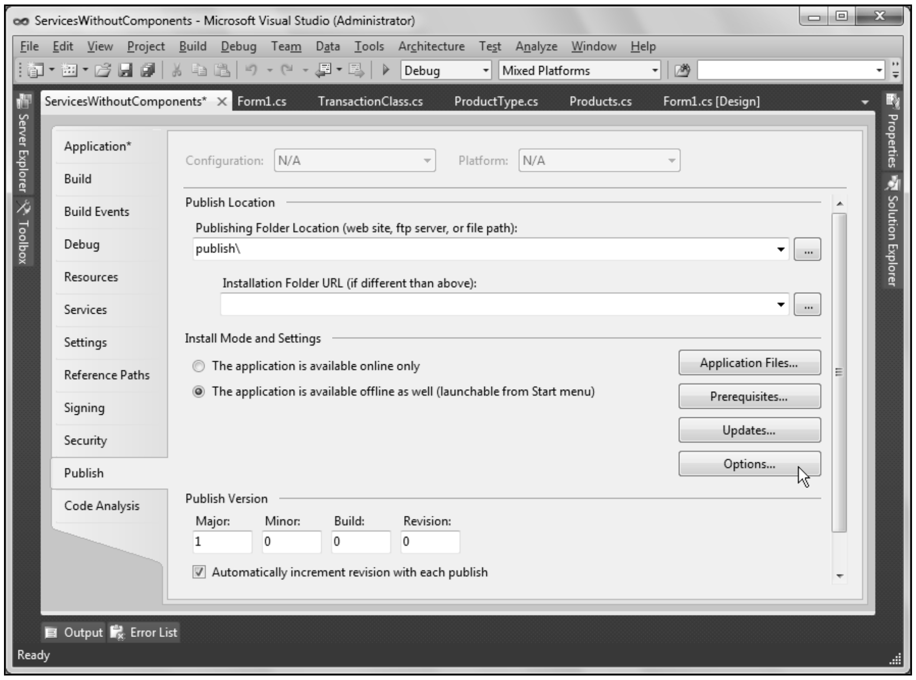
Data entered successfully

Figure 8.37: Using the ServiceDomain and ServiceConfig Classes

Using ClickOnce Deployment

You need to publish the application to a Web page, a network file share, or a CD to deploy an application using ClickOnce. You can publish an application by using the Publish Wizard. As an example, let's publish the `ServicesWithoutComponents` application (created in the preceding section) to a Web page using the Publish Wizard. Perform the following steps to publish the application:

1. Right-click the `ServicesWithoutComponents` project in Solution Explorer and select the Properties option from the context menu. The Project Designer appears, as shown in Figure 8.38:



Configuration: N/A Platform: N/A

Publish Location

Publishing Folder Location (web site, ftp server, or file path):
publish\

Installation Folder URL (if different than above):

Install Mode and Settings

( ) The application is available online only
(●) The application is available offline as well (launchable from Start menu)

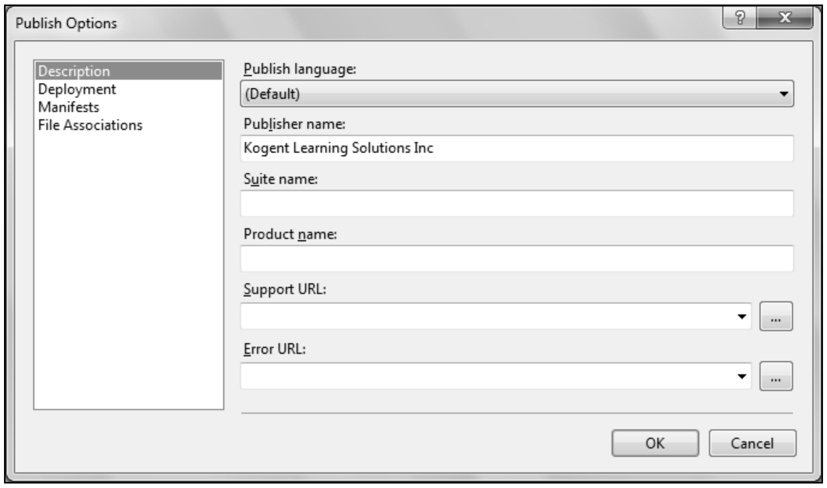
Publish Version

Major: 1 Minor: 0 Build: 0 Revision: 0

☑ Automatically increment revision with each publish

Figure 8.38: Showing the Project Designer

2. Select the Publish tab and click the Options button (Figure 8.38) to open the Publish Options dialog box, as shown in Figure 8.39:



Publish language: (Default)

Publisher name: Kogent Learning Solutions Inc

Suite name:

Product name:

Support URL:

Error URL:

Figure 8.39: Showing the Publish Options Dialog Box

3. Enter the name of the publisher in the Publisher name text box of the Publish Options dialog box (Figure 8.39). In this case, we have entered Kogent Learning Solutions Inc.
4. Select Deployment from the list shown at the left side of the Publish Options dialog box and enter a name for the Web page to be published in the Deployment web page text box. In this case, we have entered Publish.htm (Figure 8.40).

5. Select the Automatically generate deployment web page after every publish check box, as shown in Figure 8.40:

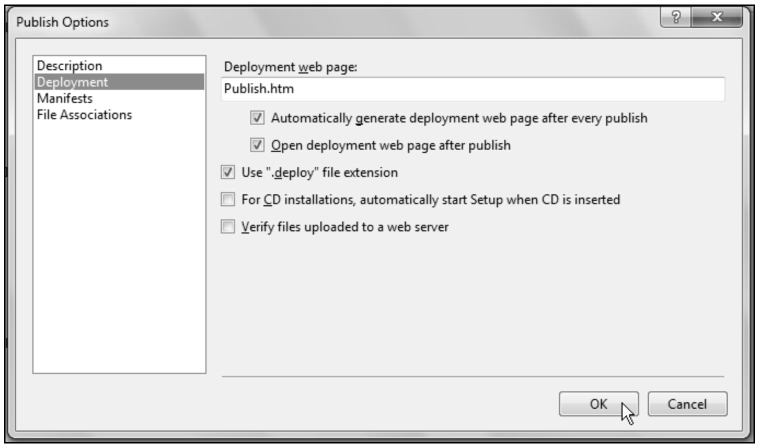
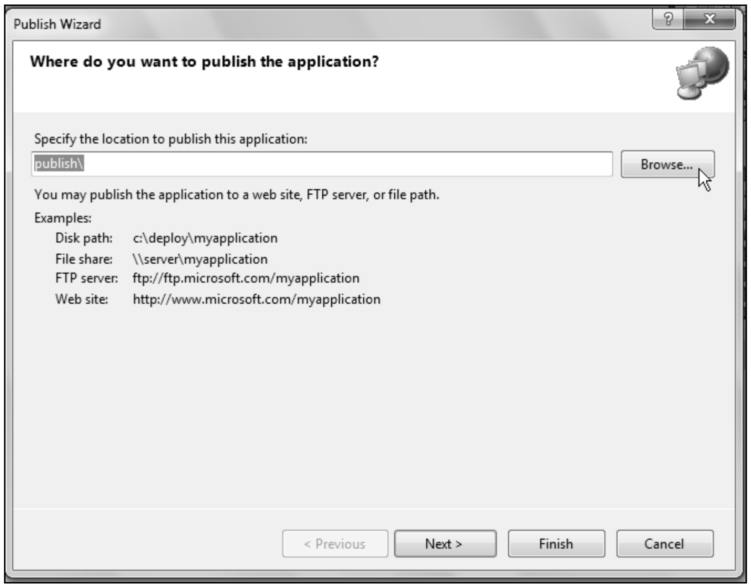

Publish Options
Description
Deployment
Manifests
File Associations
Deployment web page:
Publish.htm
☑ Automatically generate deployment web page after every publish
☑ Open deployment web page after publish
☑ Use ".deploy" file extension
☐ For CD installations, automatically start Setup when CD is inserted
☐ Verify files uploaded to a web server
OK
Cancel

Figure 8.40: Changing the Deployment Options in the Publish Options Dialog Box

6. Click the OK button to close the Publish Options dialog box.
7. Right-click the name of the ServicesWithoutComponents project in Solution Explorer and select the Publish option from the context menu that appears. This displays the first page of the Publish Wizard, as shown in Figure 8.41:


Publish Wizard

Where do you want to publish the application?

Specify the location to publish this application:

publish\

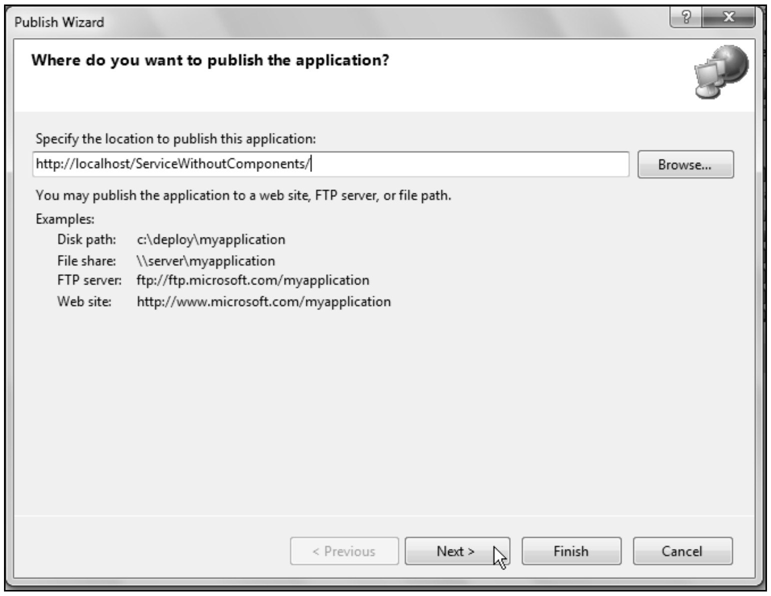
You may publish the application to a web site, FTP server, or file path.

Examples:

Disk path:	c:\deploy\myapplication
File share:	\\server\myapplication
FTP server:	ftp://ftp.microsoft.com/myapplication
Web site:	http://www.microsoft.com/myapplication

Figure 8.41: Specifying the Location to Publish the Application

8. Specify the location where you want to publish your application, such as a website, a network file share, or a CD. In our case, we are publishing the application to the Web page, so we enter `http://localhost/ServiceWithoutComponents/` as the location for the website, as shown in Figure 8.42:


Publish Wizard

Where do you want to publish the application?

Specify the location to publish this application:

http://localhost/ServiceWithoutComponents/

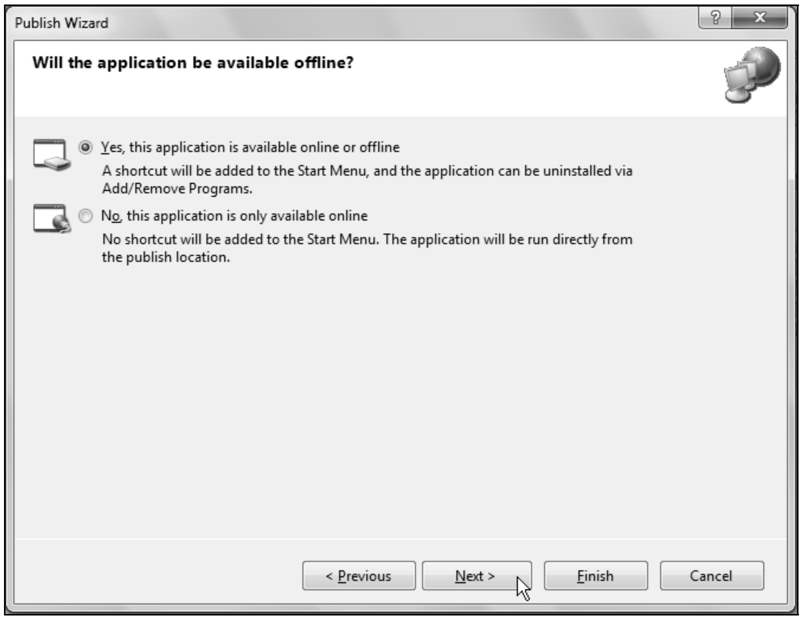
You may publish the application to a web site, FTP server, or file path.

Examples:

Disk path:	c:\deploy\myapplication
File share:	\\server\myapplication
FTP server:	ftp://ftp.microsoft.com/myapplication
Web site:	http://www.microsoft.com/myapplication

Figure 8.42: Showing the Location for the Website

9. Click the Next button. In the next page of the Publish Wizard you need to specify whether your application is available offline or not, as shown in Figure 8.43:


Publish Wizard

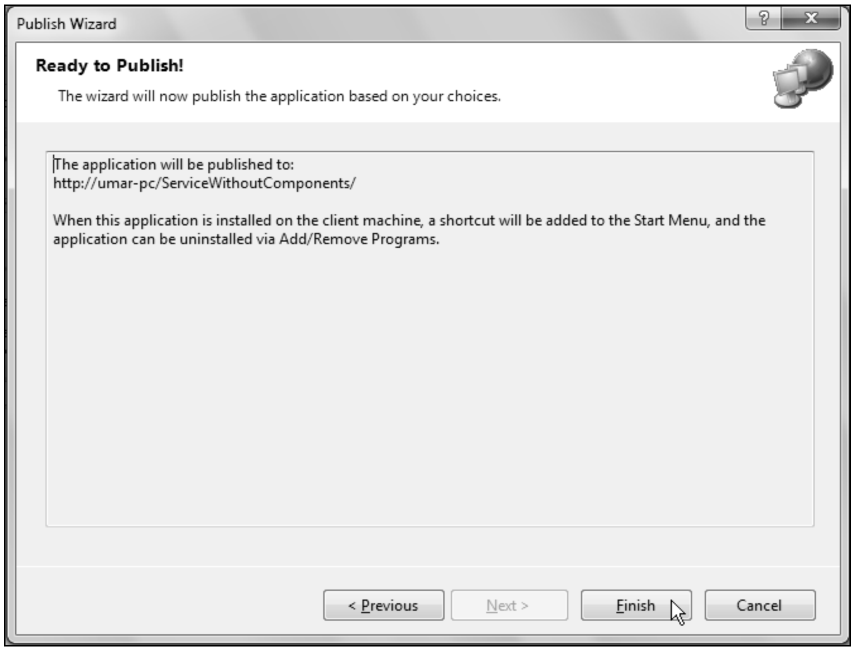
Will the application be available offline?

(●) Yes, this application is available online or offline
A shortcut will be added to the Start Menu, and the application can be uninstalled via Add/Remove Programs.

( ) No, this application is only available online
No shortcut will be added to the Start Menu. The application will be run directly from the publish location.

Figure 8.43: Selecting the Mode of the Application

10. Select the Yes, this application is available online or offline radio button and click the Next button to display the next page of the Publish Wizard. This page displays the message that the application is ready to publish, as shown in Figure 8.44:



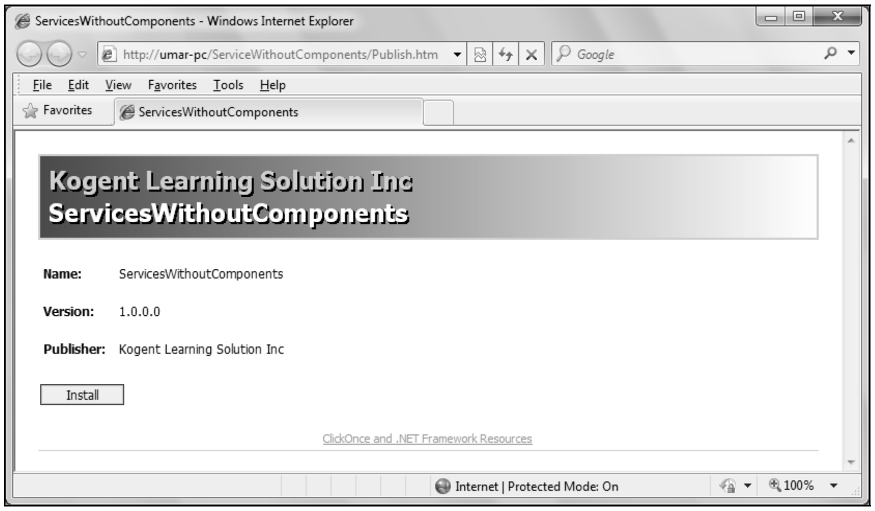
The wizard will now publish the application based on your choices.

The application will be published to:
http://umar-pc/ServiceWithoutComponents/

When this application is installed on the client machine, a shortcut will be added to the Start Menu, and the application can be uninstalled via Add/Remove Programs.

Figure 8.44: Showing the Final Page of the Publish Wizard

11. Click the Finish button to complete the Publish Wizard. This builds your project, publishes the application to the location you specified, and opens the `publish.htm` file in the browser, as shown in Figure 8.45:


ServicesWithoutComponents - Windows Internet Explorer
http://umar-pc/ServiceWithoutComponents/Publish.htm
Google
File Edit View Favorites Tools Help
Favorites
ServicesWithoutComponents
Kogent Learning Solution Inc
ServicesWithoutComponents
Install
ClickOnce and .NET Framework Resources
Internet | Protected Mode: On
100%

Name: ServicesWithoutComponents

Version: 1.0.0.0

Publisher: Kogent Learning Solution Inc

Figure 8.45: Showing the Publish.htm File

12. Click the Install button to install the `ServicesWithoutComponents` application.

After the ClickOnce application is installed and executed, an icon of the ClickOnce application is displayed on the Start menu and an entry is added to the Programs and Features option in the Control Panel. Now, you can open the ClickOnce application directly from the Start menu and can also uninstall the application using the Control Panel.

Extracting WSDL for a Remote Object

WSDL describes a Web service, i.e., it specifies the format and grammar for a Web service. You can easily get the details of the interfaces used in a Web service through a WSDL file. Web services can be exposed as remote objects with the help of WSDL interfaces. A remote object is used for distributed computing. Let's now see how to extract a WSDL file from a remote object.

First, let's create a remote object named `MathOperation` (available in the CD-ROM in `the HTTP` folder), which is a Class Library application containing a class named `ArithmeticOperation`. This class must be derived from the `System.MarshalByRefObject` class to make an object a remote object. Add four methods—`Add`, `Subtract`, `Multiply`, and `Divide`—that are called from the client. All these methods return a string to the class, `ArithmeticOperation`. Add the code, shown in Listing 8.9, in the Class1.cs file to create the remote object:

Listing 8.9: Showing the Code for the Class1.cs File

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace MathOperation
{
    public class ArithmeticOperation : System.MarshalByRefObject
    {
        int result;
        public ArithmeticOperation()
        {
            Console.WriteLine("Math operation started.");
        }
        public string Add(int num1, int num2)
        {
            result = num1 + num2;
            return "Addition of a and b : " + result.ToString();
        }
        public string Subtract(int num1, int num2)
        {
            result = num1 - num2;
            return "Subtraction of a and b : " + result.ToString();
        }
        public string Multiply(int num1, int num2)
        {
            result = num1 * num2;
            return "Multiplication of a and b : " + result.ToString();
        }
        public string Divide(int num1, int num2)
        {
            result = num1 / num2;
            return "Division of a and b : " + result.ToString();
        }
    }
}
```

To create a server application, let's create a new C# Console application named `Server` (also available in the CD-ROM in the `HTTP` folder). You need to add a reference of the `System.Runtime.Remoting` assembly to use the `HttpChannel` class. In addition, add a reference of the `MathOperation` class that we have created earlier.

Add the code, shown in Listing 8.10, in the Program.cs file of the Server application:

Listing 8.10: Showing the Code for the Program.cs File of the Server Application

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
```

```
    using System.Runtime.Remoting.Channels.Http;
    using MathOperation;
    namespace Server
    {
        public class ServerClass {
            public static void Main(string[] args) {
                HttpChannel channel = new HttpChannel(9932);
                ChannelServices.RegisterChannel(channel, false);
                RemotingConfiguration.ApplicationName = "MathOperation";

RemotingConfiguration.RegisterWellKnownServiceType(typeof(MathOperation.ArithmeticOperation),
"ArithmeticOperation", WellKnownObjectMode.SingleCall);
                Console.WriteLine("Server started.");
                Console.ReadLine();
            }
        }
    }
```

In Listing 8.10, the ServerClass class contains a Main method, in which we create an object named channel of the HttpChannel class with port number 9932. The channel object is then registered with the ChannelServices class to make the channel object available for use to the remote objects. The second parameter value of the registerChannel method is false, which disables security. The RemotingConfiguration.RegisterWellKnownServiceType method is used to register the remote object type. This method takes three parameters—the type of the remote object class, the client Uniform Resource Identifier (URI), and the mode specified. The WellKnownObjectMode.SingleCall mode creates a new object for every method call.

To create a client application, create a new C# Console application named `Client` (also available in the CD-ROM in `the HTTP` folder). Add a reference of the `System.Runtime.Remoting` assembly to use the `HttpChannel` class and also add a reference of the `MathOperation` class to the `Client` application. Now, perform the following steps to create the client application:

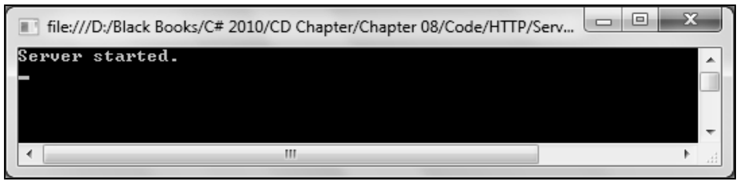
Create an object named obj of the `HttpChannel` class, which is registered in the `ChannelServices` class, in the `Client` application. The default constructor of the `HttpChannel` class is used to select a free port. The `Activator` class then returns a proxy object of the Server application to the remote object. The code for the `Client` class is given in Listing 8.11:

Listing 8.11: Showing the Code for the Program.cs File of the Client Application

```
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.Runtime;
    using System.Runtime.Remoting;
    using System.Runtime.Remoting.Channels;
    using System.Runtime.Remoting.Channels.Http;
    using MathOperation;
    namespace Client
    {
     public class Program {
             public static void Main(string[] args) {
                     ChannelServices.RegisterChannel(new HttpChannel(), false);
                     ArithmeticOperation obj =
(ArithmeticOperation)Activator.GetObject(typeof(ArithmeticOperation),"http://localhost:9932/A
rithmeticOperation");
                     if (obj == null)
                     {
                             Console.WriteLine("Unable to get remote reference");
                     }
                     else
                     {
                             char b;
                     do {
                                     Console.WriteLine("1. Addition");
                                     Console.WriteLine("2. Subtraction");
                                     Console.WriteLine("3. Multiplication");
                                     Console.WriteLine("4. Division");
```

```
                                        Console.WriteLine("Enter your choice :");
                                        int choice = int.Parse(Console.ReadLine());
                                        Console.WriteLine("Enter value for a :");
                                        int num1 = int.Parse(Console.ReadLine());
                                        Console.WriteLine("Enter value for b :");
                                        int num2 = int.Parse(Console.ReadLine());
                                        string msg;
                                        switch (choice) {
                                                case 1:
                                                msg= obj.Add(num1, num2);
                                                Console.WriteLine(msg);
                                                break;
                                                case 2:
                                                msg = obj.Subtract(num1, num2);
                                                Console.WriteLine(msg);
                                                break;
                                                case 3:
                                                msg = obj.Multiply(num1, num2);
                                                Console.WriteLine(msg);
                                                break;
                                                case 4:
                                                msg = obj.Divide(num1, num2);
                                                Console.WriteLine(msg);
                                                break;
                                                default:
                                                Console.WriteLine("no match found");
                                                break;
                                        }
                                        Console.WriteLine("Do you want to continue (Y/N)? :");
                                        b= Convert.ToChar (Console.ReadLine());
                                }
                                while (b  == Convert.ToChar("Y"));
                        }
                }
        }
    }
```

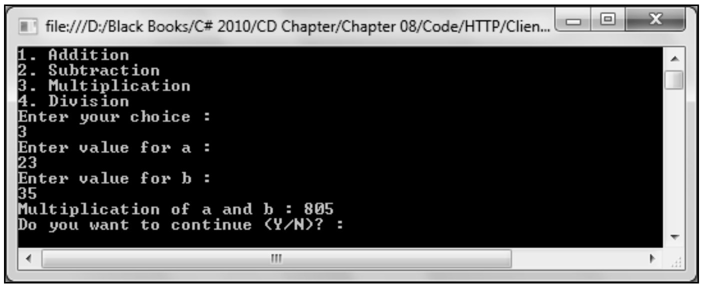
2. Run the Server application first and then the Client application. When you start the server, a message Server started is displayed, as shown in Figure 8.46:



```
file:///D:/Black Books/C# 2010/CD Chapter/Chapter 08/Code/HTTP/Serv...
Server started.
_
```

Figure 8.46: Showing the Server Console Window

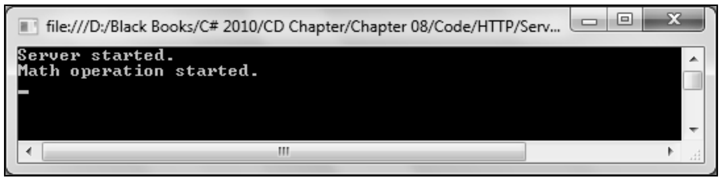
3. Run the Client application. In the Client Console window, you are provided with different options, such as Addition, Subtraction, Multiplication, and Division.
4. Choose the operation you want to perform and then enter two integer values for the a and b variables. On the basis of the choice made, the result is displayed on the Client Console, as shown in Figure 8.47:



```
file:///D:/Black Books/C# 2010/CD Chapter/Chapter 08/Code/HTTP/Clien...
1. Addition
2. Subtraction
3. Multiplication
4. Division
Enter your choice :
3
Enter value for a :
23
Enter value for b :
35
Multiplication of a and b : 805
Do you want to continue (Y/N)? :
```

Figure 8.47: Showing the Output Window of the Client

The Server Console window automatically shows the result after running the Client Console application. The output of the Server Console window is shown in Figure 8.48:


file:///D:/Black Books/C# 2010/CD Chapter/Chapter 08/Code/HTTP/Serv...

```
Server started.
Math operation started.
_
```

Figure 8.48: Showing the Output Window of the Server

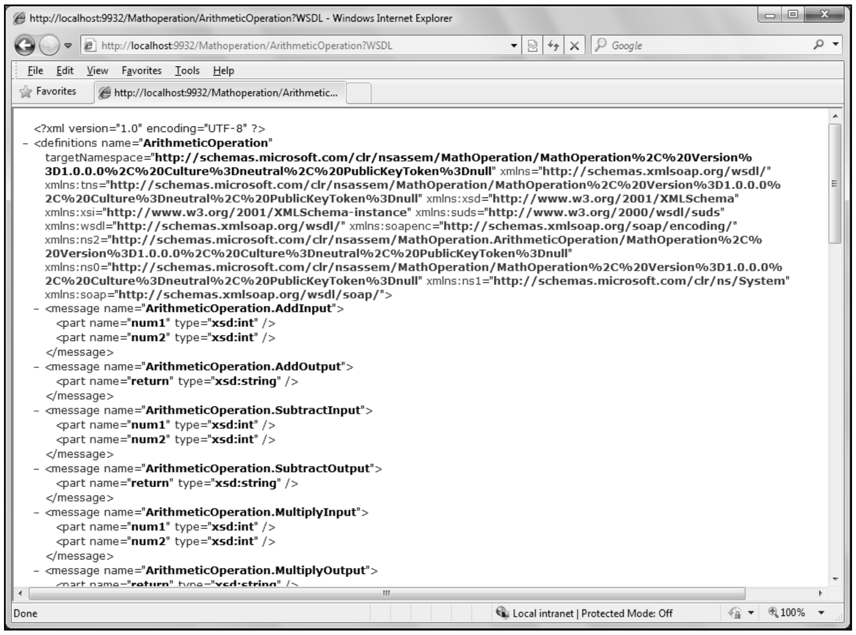
5. To extract a WSDL file for the remote object, use the following URL format:

```
http://<computer name>:<Port Number>/<VirtualDirectory>/<ObjectURI>?wsdl
```

6. To view the WSDL file, open the following URL (the server needs to be started before that):

```
http://localhost:9932/Mathoperation/ArithmeticOperation?WSDL
```

The preceding URL displays the WSDL file, as shown in Figure 8.49:


http://localhost:9932/Mathoperation/ArithmeticOperation?WSDL - Windows Internet Explorer
http://localhost:9932/Mathoperation/ArithmeticOperation?WSDL
Google
File Edit View Favorites Tools Help
Favorites
http://localhost:9932/Mathoperation/Arithmetic...
Done
Local intranet | Protected Mode: Off
100%

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="ArithmeticOperation"
    targetNamespace="http://schemas.microsoft.com/clr/nsassem/MathOperation/MathOperation%2C%20Version%
    3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull" xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://schemas.microsoft.com/clr/nsassem/MathOperation/MathOperation%2C%20Version%3D1.0.0.0%
    2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:suds="http://www.w3.org/2000/wsdl/suds"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:ns2="http://schemas.microsoft.com/clr/nsassem/MathOperation.ArithmeticOperation/MathOperation%2C%
    20Version%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull"
    xmlns:ns0="http://schemas.microsoft.com/clr/nsassem/MathOperation/MathOperation%2C%20Version%3D1.0.0.0%
    2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull" xmlns:ns1="http://schemas.microsoft.com/clr/ns/System"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  - <message name="ArithmeticOperation.AddInput">
      <part name="num1" type="xsd:int" />
      <part name="num2" type="xsd:int" />
    </message>
  - <message name="ArithmeticOperation.AddOutput">
      <part name="return" type="xsd:string" />
    </message>
  - <message name="ArithmeticOperation.SubtractInput">
      <part name="num1" type="xsd:int" />
      <part name="num2" type="xsd:int" />
    </message>
  - <message name="ArithmeticOperation.SubtractOutput">
      <part name="return" type="xsd:string" />
    </message>
  - <message name="ArithmeticOperation.MultiplyInput">
      <part name="num1" type="xsd:int" />
      <part name="num2" type="xsd:int" />
    </message>
  - <message name="ArithmeticOperation.MultiplyOutput">
      <part name="return" type="xsd:string" />
```

Figure 8.49: Showing the WSDL File of the Remote Object

Let's now summarize the main topics discussed in this chapter.

Summary

In this chapter, you have learned about Enterprise Services, automated transactions, distributed transactions, contexts, object pooling, queued components, and loosely coupled events. You have also learned to create a simple COM+ application, and then a simple Client application that accesses the COM+ application. Next, you have learned to create components without inheriting the `ServicedComponent` class. Further, you have learned about automatic deployment, manual deployment, and ClickOnce deployment. In addition, you have learned to expose a COM+ application as WCF service, and export object proxies with the help of the Export wizard.

In the next chapter, you learn about application globalization.