A

Creating Business Objects

Component Object Model (COM+) services (which is an extension of COM), when used with .NET are called Enterprise Services. COM+ technology facilitates in writing component-based applications and also facilitates interoperability between components. Enterprise Services is the name given to the Microsoft application server technology that provides services for distributed solutions. In the .NET technology (hereafter .NET), COM+ services are referred to as Enterprise Services that help developers to build and develop robust server applications. Enterprise Services provide services, such as role-based security, Microsoft's queued components, transactions, loosely coupled events, and object pooling.

You must be wondering now as to where do we need to use Enterprise Services. A business application is normally divided into three layers—presentation, business, and data services layers. 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. The data service layer interacts with the databases. You can use Enterprise Services both in case of business and data service layers.

In this appendix, we cover some concepts related to the COM+ technology.

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 intercepts a method when an object of one context calls the object of another context. The proxy provides the necessary services to the COM+ runtime to fulfill the object's needs, such as call serialization, automatic transaction management, and the rest.

Transactions

A transaction can be defined as a set of executable 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. Thus, transactions prevent any loss of data when the system shuts down due to power failure or system failure.

Transactions are especially used in situations where databases are used and where the chances of failures are considerably high. Transactions bring an application back to its original state, that is, to a state it was before making any changes to the application. Thus, transactions help in maintaining a valid state.

ACID Properties

ACID properties are used with respect to databases. The acronym ACID stands for Atomicity, Consistency, Isolation, and Durability. These properties were introduced with the purpose of safe sharing of data and keeping a check on inaccurate data from entering databases. When huge number of transactions take place simultaneously for purchasing the same type of product, it becomes difficult to maintain accuracy and consistency of data if these ACID properties are not used. Implementing ACID properties ensures error-free transactions.

As you have seen what ACID properties imply, let's now have an insight into each property one by one.

Atomicity

Atomicity ensures that if any update occurs in a database, then either all of the transactions execute successfully or none of them executes. This means either the update operation will successfully commit or it will abort completely. This property ensures that if the transaction does not commit successfully, the system will return back to its original state.

Consistency

Consistency ensures that any changes made in the database are valid and consistent before and after the update is committed successfully. If a transaction executes successfully, the system returns to a state where all the changes relating to an update are successfully made wherever required, whereas if the transaction does not execute successfully, the transaction is rolled back and the system will return to its original valid state, thus ensuring complete consistency.

Isolation

If multiple transactions occur simultaneously (called as concurrent transactions), isolation ensures that the transactions are isolated from each other until each transaction executes completely. For instance, consider a situation in which two transactions are accessing the same resource, performing the same task, and executing at the same time. In this situation, if isolation is not implemented, erroneous data is stored in the database. Thus, to prevent the erroneous data from getting stored in the database, transaction isolation is used. Thus, transaction isolation ensures that each transaction is isolated from one another and considers a particular transaction to be the only one that is making use of the resources. When a transaction occurs, then at the time of transaction execution the system may not be in a valid state. Transaction isolation ensures that the transaction is isolated and the system is in a valid state when a transaction accesses the resources. In case, if a transaction accesses the resources and is not in isolation, it might access the resources when the system is not in a valid state. Isolation thus prevents access to inconsistent data.

Durability

The last property is durability, which ensures that in case of any power failure, server crash, or any kind of failure, which affects the system, then the system will be restored with the updates of the last successfully committed transaction after reboot. Durability ensures that the changes made to the system are permanent after updating.

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 of Microsoft, Oracle, and the rest.

All the Enterprise Services transactions are coordinated by the Distributed Transaction Coordinator (DTC) and thus 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 contains two phases, the prepare phase and the commit phase. When data is updated, first a prepare phase occurs, in which the transactions must complete the operations successfully. In case, if any

one of the transactions cancels or aborts, a rollback occurs for all the participants. In the commit phase, all the data is updated in the database, if the prepare phase is completed successfully.

Automated Transactions

You need not pass any argument as a transaction to any method while using automated transactions. Instead of that an attribute `[Transaction]` is applied to the class. The `[Transaction]` attribute with different options, such as `Required`, `Supported`, `RequiresNew`, `NotSupported`, and the rest, can be used to specify whether a transaction is required or not. When you specify a class with the `Required` option, a transaction is automatically created when a method starts. Automated transactions are particularly of great advantage when you develop a complex object model.

Table A.1 displays the attribute values used with objects in an automatic transaction model:

Table A.1: 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, and 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`	Means that the component will not participate in a transaction irrespective of whether the caller has a transaction. The code to specify the `Supported` attribute is as follows: `[Transaction(TransactionOption.NotSupported)].`
`Required`	Specifies that the object must have a transaction. If a transaction is already created, the object will run within the scope of the transaction. If there is no transaction created, it will create a new transaction. The code to specify the `Supported` attribute is as follows: `[Transaction(TransactionOption.Required)].`
`RequiresNew`	Creates a new transaction. The component never participates in the same transaction as a caller. The code to specify the `Supported` attribute is as follows: `[Transaction(TransactionOption.RequiresNew)].`

Object Pooling

Object pooling is a service provided by Enterprise Services. 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 and the moment the application is closed, the object of the component is destroyed. Object pooling maintains a list of active objects of components in a pool, which is handled through the pool manager. 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. Thus, object pooling enables increased performance and scalability of applications, and managing 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, database connections, and the rest
- Makes fast access possible in case of re-activating the objects that are enabled just in time

- Facilitates optimum utilization of the hardware resources since pool configuration changes with the hardware configuration

Queued Components

Queuing in simple terms means communication between the sender and the receiver without the need of any connection. It means that even if the sender and receiver are not present or are not connected, the processing occurs without any problem. The messages in the queue are held until the user is ready to accept the messages. The messages in a queue are saved so that they can be retrieved later when needed.

Queued components are a service provided by Enterprise Services. This feature is based on Microsoft Message Queuing Service (MSMQ). 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. This feature helps in easy execution of the components. Queued components and message queuing help 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 A.1 explains the queuing of messages:

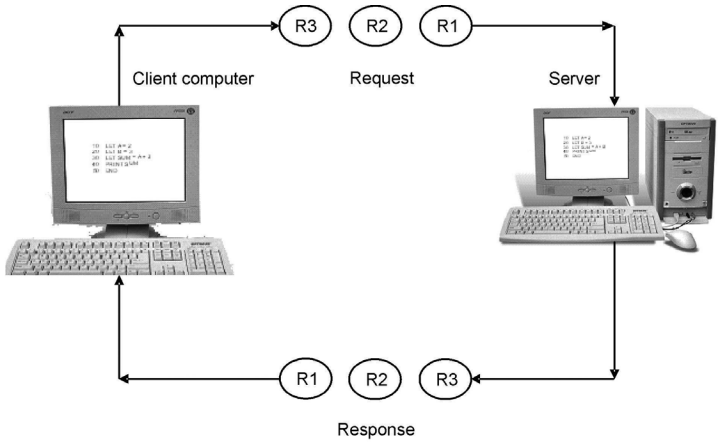

Figure A.1: Queuing of Messages

Figure A.2 explains the queued component architecture:

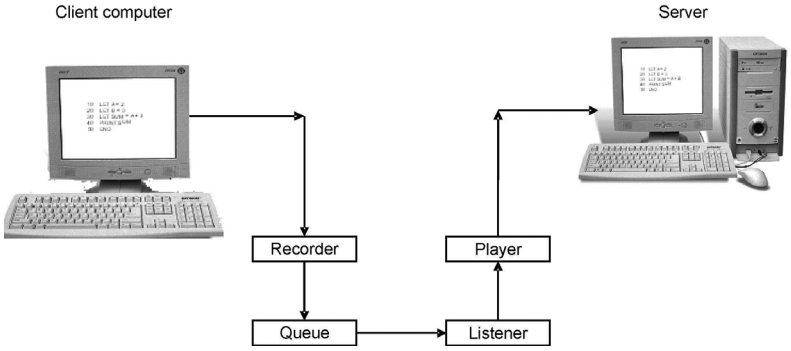

Figure A.2: Queued Component Architecture

Loosely Coupled Events

A loosely coupled event (LCE) helps in implementing COM+ services between the client and the server. COM+ event services are implemented with the help of the following four main entities:

- **Event Publisher**—Registers the events that might be needed by subscribers. Subscribers request for the event. The event information is stored in the COM+ catalog. You can find out subscribers who need specific events. However, it is not necessary for the publisher to know subscribers. Similarly, subscribers can use the events in the COM+ catalog without knowing the publisher.
- **Event Class**—Acts as a mediator between the publisher and the subscriber as direct communication between the publisher and the subscriber 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 a subscription for an event. The LCE service passes events to Event Subscribers who request for the events. The Event Subscriber is a client application.
- **Event Client**—Requests for the events. The Event Client is a client application.

Figure A.3 shows the functioning of the loosely coupled event services:

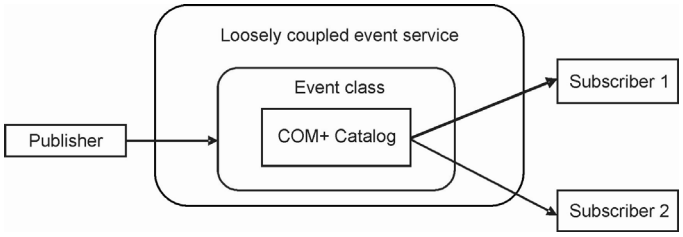

Figure A.3: Loosely Coupled Event Service

The ServicedComponent Class

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

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

Table A.2 lists the methods of the `ServicedComponent` class:

Table A.2: Protected Methods of the ServicedComponent Class	
Protected Method	**Description**
`Activate`	Called when an object is created from a pool. This method is overridden to customize the initialization code for the objects.
`Deactivate`	Called when an object is to be deactivated. This method is overridden to customize the finalization code for the objects.
`CanBePooled`	Called before the object is placed inside the pool. This method is overridden to indicate whether or not the object is placed into the pool.
`Construct`	Called after the constructor is called which takes a string as a parameter. This method is overridden to use the constructor string value.

These preceding methods can be overridden in their derived classes.

Setting Application Attributes

Enterprise Services provide some `Application` attributes that are needed to build Enterprise Services applications. Table A.3 lists the `Application` attributes with their description:

Table A.3: List of Application Attributes	
Protected Method	**Description**
`ApplicationAccessControl`	Turns off the security and allows any user to use a component. The following code is used to set this attribute: `[assembly: ApplicationAccessControl(false)]`.

Table A.3: List of Application Attributes	
Protected Method	**Description**
`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. The following code is used to set this attribute: `[assembly: ApplicationActivation(ActivationOption.Server)].`
`ApplicationID`	Sets an ID (GUID) for the assembly. The following code is used 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). The following code is used to set this attribute: `[assembly: ApplicationName("Component")].`
`ApplicationQueuing`	Supports queuing for the assembly. The following code is used to set this attribute: `[assembly: ApplicationQueuing].`

The ServiceConfig Class

Use of transactions and synchronization services makes it easier to create a COM+ context without inheriting from the `ServicedComponent` base class. The classes used for creating services without components are the `ServiceConfig` and the `ServiceDomain` classes. The `ServiceConfig` class configures the context. The `ServiceDomain` class creates a context. Here is the class hierarchy for the `ServiceConfig` class:

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

Table A.4 displays some properties of the `ServiceConfig` class:

Table A.4: Properties of the ServiceConfig Class	
Property	**Description**
`Inheritance`	Helps you in specifying whether the new context created should derive from the existing context, or a new context should be created. By default, the new context derives from the existing context with the `InheritanceOption.Inherit value`. If the value is `InheritanceOption.Ignore`, then 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 describing the transaction.
`TransactionTimeout`	Obtains or sets the transaction time-out for a new transaction.

.NET and COM Technologies

The COM technology was introduced first and then it was followed by .NET. COM contains components that can be written by using different programming languages. In the COM technology, components can be used within the same process, across different processes, and also across the network. These facilities are also available in .NET. COM components are more complex and are not extensible enough. .NET introduced new concepts to address these shortcomings and makes it easier to build components.

There is no doubt that .NET is much powerful than the COM technology in building components, but that is not the end for the COM technology. The COM technology will stay and can be used with.NET. However, the problems associated with the COM technology remain as it is when the COM technology is interoperated with

.NET applications. The most frequently used concepts in both the COM and .NET technologies are stated as follows:

- **Metadata**—In the COM technology, all the details of a component are stored inside the type library. The type library consists of names and ids of interfaces, methods, and arguments. .NET contains this information inside the assembly. The problem that the COM technology faced was that the type library could not be extended, whereas in .NET the metadata can be extended using the custom attributes. In .NET, the metadata is used to describe information about the assembly, such as identity, types, security permissions, and dependent assemblies inside the assembly.
- **Freeing memory**—.NET uses the Garbage Collector to release the memory, whereas in the case of the COM technology it depends on reference counts to free the memory. COM uses the `IUnknown` interface, which must be implemented by every COM object. This interface provides three methods out of which two are related to reference counts. The `AddRef` method increments the reference count and must be called by the client if another interface pointer is required. The `Release` method decrements the reference count, and when the reference count is set to `0`, it means that the object needs to be destroyed and the memory released.
- **Method binding**—There are mainly two types of binding that a client generally uses with method binding, early and late binding. Late binding means the method to bind is looked for at runtime, whereas early binding binds the method at design time. Late binding is implemented in .NET using the `System.Reflection` namespace, whereas in the COM technology the `IDispatch` and `Dual` interfaces are used for late binding.

  In the COM technology, early binding can be implemented in two different ways, one way is to use the `vtable` binding which uses `vtable` with custom and dual interfaces, and the second way, also known as id binding, uses the dispatch id which is stored in the client code. At runtime, a call is made to the `Invoke` method to call the dispatch id.
- **Registration**—.NET considers private and shared assemblies as different entities, as discussed in Appendix 34, .NET Assemblies, to maintain difference between COM and .NET technology components. All the COM components are accessed using the registry entries. These registry entries contain the CLSID from the registry and the DLL or EXE path to load the DLL or EXE of the COM component and instantiate the component. COM components can also be searched by using the prog-id that can be stated as Word. Application or by some other name.
- **Error Handling**—In .NET, errors are raised by throwing exceptions, whereas in the COM technology errors are raised by returning HRESULT values with the methods. When the HRESULT value contains `S_OK`, it means that the method was successful.

  In order to get a more detailed error message from the COM component, the `ISupportErrorInfo` interface needs to be used. A detailed error message includes an error message with a link to a help file, and the source of the error.
- **Event handling**—Event handling in .NET is possible through events and delegates. In COM events, the components need to implement the `IConnectionPointContainer` interface and some connection point objects (CPOs) that implement the interface `IConnectionPoint`. An outgoing interface `ICompletedEvents` is also defined by the component. The sink object within the client which is also a COM object must implement the outgoing interface. At runtime, the client requests the server for the `IConnectionPointContainer` interface. The client through this interface asks for a CPO with the `FindConnectionPoint` method to get a pointer to the `IConnectionPoint interface`. Through this interface, the client calls the `Advise` method where a pointer to the sink object is passed to the server. In return, the component calls the methods inside the sink object of the client.
- **Marshaling**—While passing data from the .NET to COM component or from the COM to .NET component, the data must be converted according to the technology used. This technique is known as marshaling. This depends on the data type of the data passed.

There are mainly two types of data types, blittable and non-blittable data types. Blittable data types are a common representation of both the COM and .NET technologies and therefore do not require any conversion when they are passed between managed and unmanaged code. Blittable data types are simple data types, such as `byte, int, long`, and classes and arrays with one dimension. Non-blittable data types require conversion. Table A.5 displays some of the non-blittable data types with the corresponding .NET data types:

Table A.5: Non-Blittable Data Types with Corresponding .NET Data Types

COM Data Type	.NET Data Type
`SAFEARRAY`	`Array`
`Variant`	`Object`
`BSTR`	`String`
`IUnknown*, IDispatch*`	`Object`

Accessing COM Components from a .NET Client

Usually, when you try to access a COM component from a .NET Client, you cannot communicate with the COM component as the interface exposed by the COM component is not readable for the .NET client. Thus, the COM component is required to be wrapped in such a way that the .NET component can understand the COM communication. This wrapping of COM component is called Runtime Callable Wrapper (RCW). RCW is provided by .NET to wrap COM components and enable communication between the .NET with COM components. RCW can also be generated with the help of the Type Library Importer (`TlbImp.exe`) utility. Figure A.4 shows the working of RCW:

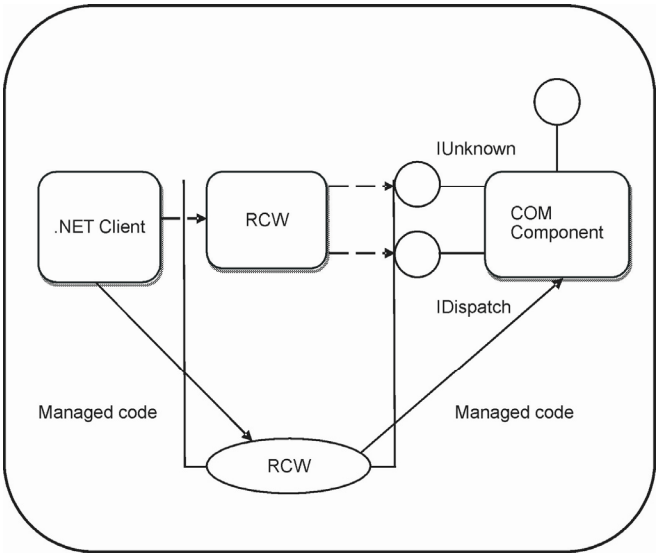

Figure A.4: Working of RCW

Accessing a .NET Component from a COM Client

Usually, when you try to access a .NET component from a COM client, it searches for the Registry entry and then starts communicating. You know that .NET communicates through objects and object-based communication is difficult for a COM client to understand. Thus, the .NET component needs to be wrapped in such a way that the COM components can understand .NET communication. This wrapping of .NET Component is called COM Callable Wrapper (CCW). CCW is provided by .NET to wrap .NET components and enable communication between the COM and .NET components. CCW can be generated with the `RegAsm.exe` utility provided by .NET. Figure A.5 shows the working of CCW:

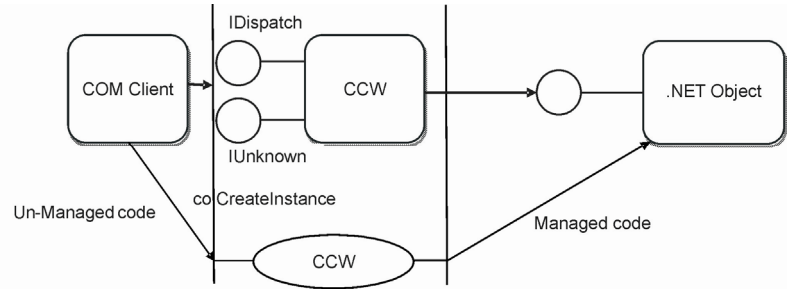

Figure A.5: Working of CCW

A Simple COM+ Application

To configure a class with Enterprise Services, we need to use the `ServicedComponent` class, which belongs to the `System.EnterpriseServices` namespace.

To understand this you need to create a simple serviced component application named `Component`. Create a class library application in Visual Studio 2010 named `Component` (available in the CD-ROM as `ComponentVB` and `ComponentCS`), since all COM+ applications must be written as library applications irrespective of whether they run in the client process or within their own process. 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. Add one more reference to the `System.Reflection` namespace in the `Class1.cs` or `Class1.vb` file to provide an organized view of the loaded types, and methods with the capacity to dynamically create and call types.

Creating a Serviced Component

The `ServicedComponent` class acts as a mediator between a client and a component. Here, the `Class1` class is used to create a serviced component class. First, rename the `Class1` class as `Component` and then create an interface with the name `IMathOperation`, which contains a method called `Add`. The `Add` method takes two integer parameters in the `Component` class. You can find the code for the `Component` class in Listing A.1:

Listing A.1: Showing the Code for the `Component` class

In VB

```
Imports System.EnterpriseServices
Imports System.Reflection

<Assembly: ApplicationName("Component")>
<Assembly: ApplicationAccessControl(False)>
<Assembly: ApplicationActivation(ActivationOption.Server, SoapVRoot:="Component")>
<Assembly: Description("Simple Serviced Component Sample")>
<Assembly: AssemblyKeyFile("Component.snk")>

Public Interface IMathOperation
    Function Add(ByVal num1 As Integer, ByVal num2 As Integer) As Integer
End Interface
<EventTrackingEnabled(True), Description("Simple Serviced Component Sample")> _
Public Class Component
    Inherits ServicedComponent
    Implements IMathOperation
    Public Sub New()
    End Sub
    Public Function Add(ByVal num1 As Integer, ByVal num2 As Integer) As Integer
     Implements IMathOperation.Add
        Return num1 + num2
    End Function
End Class
```

In C#

```
using System;
using System.Collections.Generic;
using System.Linq;
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;
        }

    }
}
```

NOTE

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

The `Component` class is derived from the `ServicedComponent` class. It also implements the `IMathOperation` interface. The `[EventTrackingEnabled]` attribute monitors the objects from the administrative tool. This attribute should be set to True since, by default, it is false. The `[Description]` attribute shows the text specified for a component in the administrative tool. A default constructor is also required to create the `ServicedComponent` class.

Here is the source code of the `ServicedComponent` class in VB:

```
<EventTrackingEnabled(True), Description("Simple Serviced Component Sample")> _
    Public Class Component
        Inherits ServicedComponent
        Implements IMathOperation
        Public Sub New()

        End Sub
```

Here is the source code of the `ServicedComponent` class in C#:

```
[EventTrackingEnabled (true)]
    [Description("Simple Serviced Component Sample")]

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

```
            }
        }
```

The `Add`function takes two integer arguments and returns the addition of these two numbers:

Here is the Add function in VB:

```
    Public Function Add(ByVal num1 As Integer, ByVal num2 As Integer) As Integer Implements
IMathOperation.Add
                Return num1 + num2
            End Function
```

Here is the Add function in C#:

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

Now, let's discuss how to develop a client application but before that a strong name needs to be created to share the assembly. Also the assembly needs to be registered using the `regasm` .NET command utility.

NOTE

Before moving further, set the ComVisible property in the AssemblyInfo.cs file to true.

Creating a Strong Name

To share the assembly, a strong name needs to be created. Such strong names can be created with the help of a tool called `sn`. Type the following line at the Visual Studio 2010 Command Prompt:

```
    sn –k Component.snk
```

After creating a strong name, add two class diagram files with the names ClassDiagram1.cd and ClassDiagram2.cd and build the application by selecting the Build→ Build Solution menu option.

Register Assembly

You can register the assembly by using the `regasm` utility from the Visual Studio 2010 Command Prompt as follows:

```
    regasm Component.dll
```

Creating a Simple Client

To use the serviced component created, build a client application. This can be built using a simple Web site called `ClientApplication` (available in the CD-ROM as `ClientApplicationVB` and `ClientApplicationCS`). Add references of the newly created `Component` namespace to the client project. Add two controls, a Button control and a Label control, of the default page of your application. On the click event of the Button control, create a new object of the `Component` class with the name `com` and invoke the `Add` function with two integer values passed into it. The `Add` function adds the two numbers and returns the result of the addition, which is displayed on the Label control. Now, add the code in the code-behind file of the `Default.aspx` page as shown in Listing A.2:

Listing A.2: Showing the code for the Code-Behind File of the `Default.aspx Page`

In VB

```
    Imports Component

    Partial Class _Default
        Inherits System.Web.UI.Page

        Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
             Handles Button1.Click
            Dim com As New Component.Component
            Label1.Text = "The addition of the two numbers is " & com.Add(10, 24)
        End Sub
    End Class
```

In C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using Component;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Component.Component com = new Component.Component();
        Label1.Text = "The addition of the two numbers is " + com.Add(10, 24);

    }
}
```

The output of the preceding listing is shown in Figure A.6:

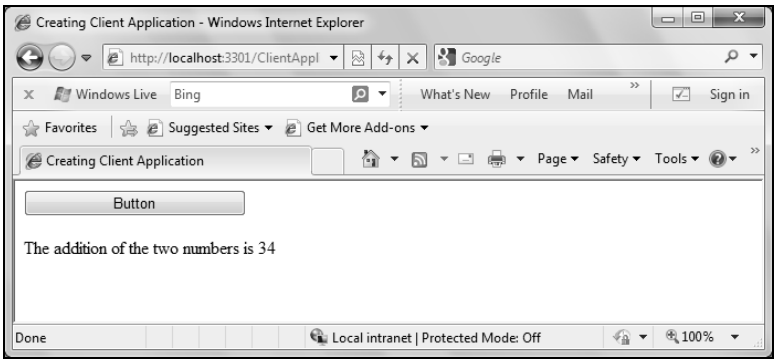

Figure A.6: Output After Running Client Application

Using Automatic Deployment

Assemblies that have serviced components must be deployed. The deployment can be either automatic or manual (through registering the assembly manually). When the client application, which uses the component, is started, 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 lacks administrative rights. In such a case, automatic deployment is possible only during development time. This feature is quite advantageous, since, during the development phase, a manual deployment after every single build operation is not required.

Using Manual Deployment

The deployment of the .NET application can be done manually using the `regsvcs.exe` utility. The command to deploy a .NET application from the Visual Studio 2010 Command Prompt is as follows:

```
regsvcs Component.dll
```

This 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.

Using the CSAT

Now that you have configured and deployed the `Component` application, perform the following steps:

1. Run the CSAT tool by typing comexp.msc in the Run dialog box and clicking the OK button, as shown in Figure A.7:

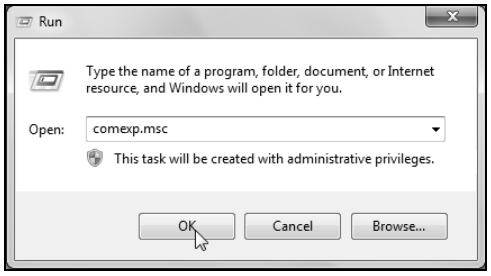

Figure A.7: Running the CSAT

This opens the Component Services Administration Tool (CSAT), where you can find all the published COM+ components, as shown in Figure A.8:

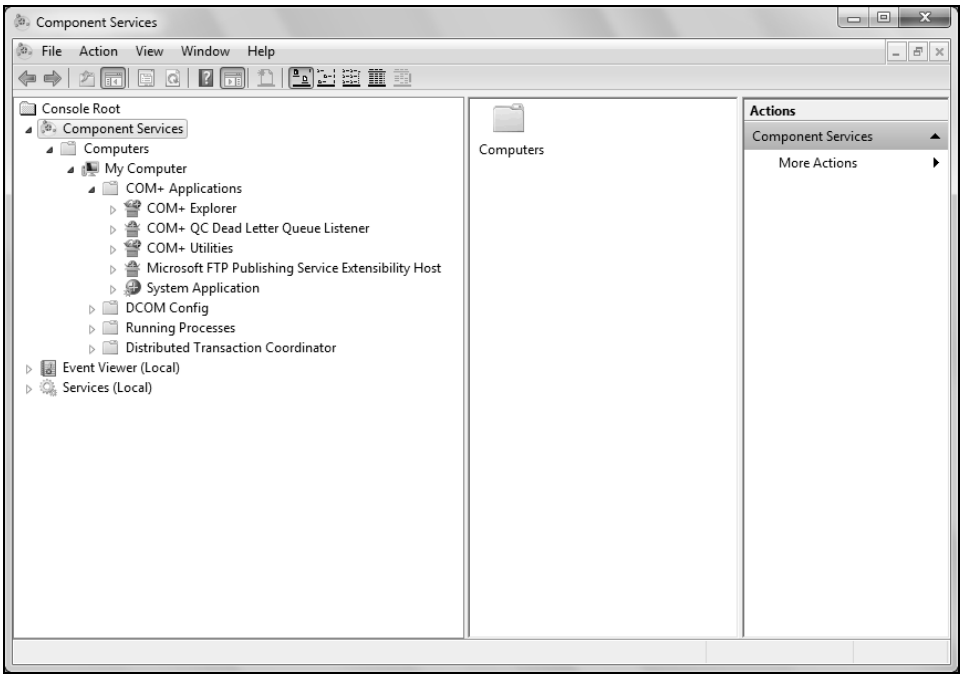

Figure A.8: Component Services Administration Tool

2. To view the properties of any component, right-click the node and select the Properties option, as shown in Figure A.9:

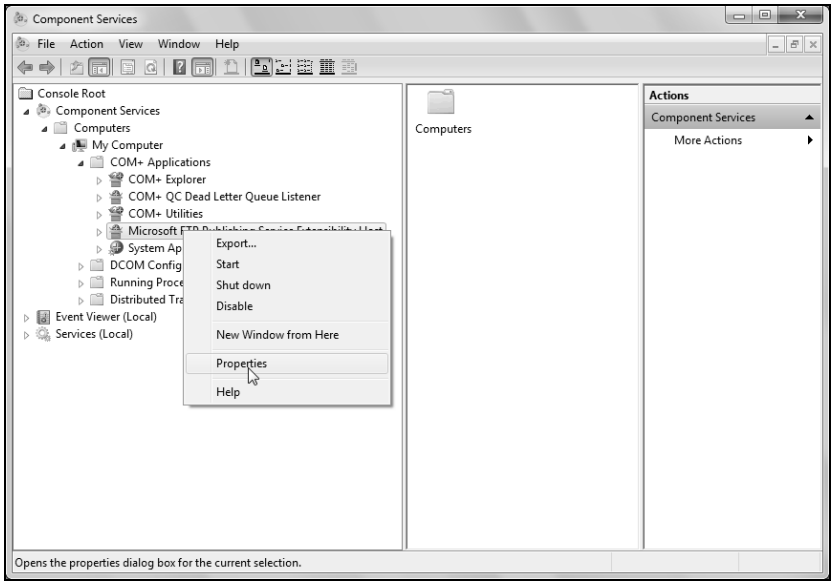

Figure A.9: Selecting the Properties Menu Item in the CSAT

This opens the Properties dialog box under the General tab, as shown in Figure A.10:

Figure A.10: General Tab of the Component Properties Dialog Box

You can see `in` the Activation tab, the application is set as a Server application because of the attribute set in the application as `[assembly: ApplicationActivation(ActivationOption.Server)]`.

You can see in the Security tab, the Enforce access checks for this application check box is not checked because of the attribute `[assembly: ApplicationAccessControl(false)]`.

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

- **Security**—The Security option facilitates in enabling or disabling access to users. In case, you enable the security, the access level is set to either the application level or the component level or the interface, or the method level. The messages sent through networks can also be encrypted. However, this affects the performance of the application.
- **Identity**—When an application is a server application, the Identity tab can be configured for the user account that uses the process that hosts the application. By default, the user selected is always the interactive user. When you are debugging the application, this setting will benefit 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**—Through the Activation option, an application can be set either as a library or as a server application. The application can be run as a Windows service or you can use Simple Object Access Protocol (SOAP) to access the application.
- **Queuing**—The Queuing option is helpful for components that use message queuing services.
- **Advanced**—With the help of the Advanced option, an application can be shut down 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**—This option 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 components developed in C++.
- **Pooling and Recycling**—Using this option an application can be configured to restart on the basis of the lifetime, memory, number of calls, and the rest, of an application.

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

3. Now, to set the configuration properties for the component, select the component in the application and click the Action→Properties menu option. This opens the Component Properties dialog box, as shown in Figure A.11:

Figure A.11: The General Tab of the Component Properties Dialog Box

You can configure the following options through this dialog box:

- **Transactions**—The Transactions tab helps to specify whether or not the component requires transaction.
- **Security**—The Security option first checks whether or not the security feature is enabled. If yes, then through this feature roles for the Component can also be specified.
- **Activation**—Object pooling can be set through the Activation tab and it assigns a construction string.
- **Concurrency**—The Concurrency option is set to either Required or Requires New, if the component is not thread-safe. Thus, at runtime only one thread is allowed at a time to access the component.

Installing and Exporting Components using CSAT

Apart from the manual and automatic deployment, you can also use the CSAT to deploy a COM+ application. Perform the following steps to do so:

1. Open the CSAT and right-click the COM+ Applications node under My Computer and select the New→Application option from the context menu, as shown in Figure A.12:

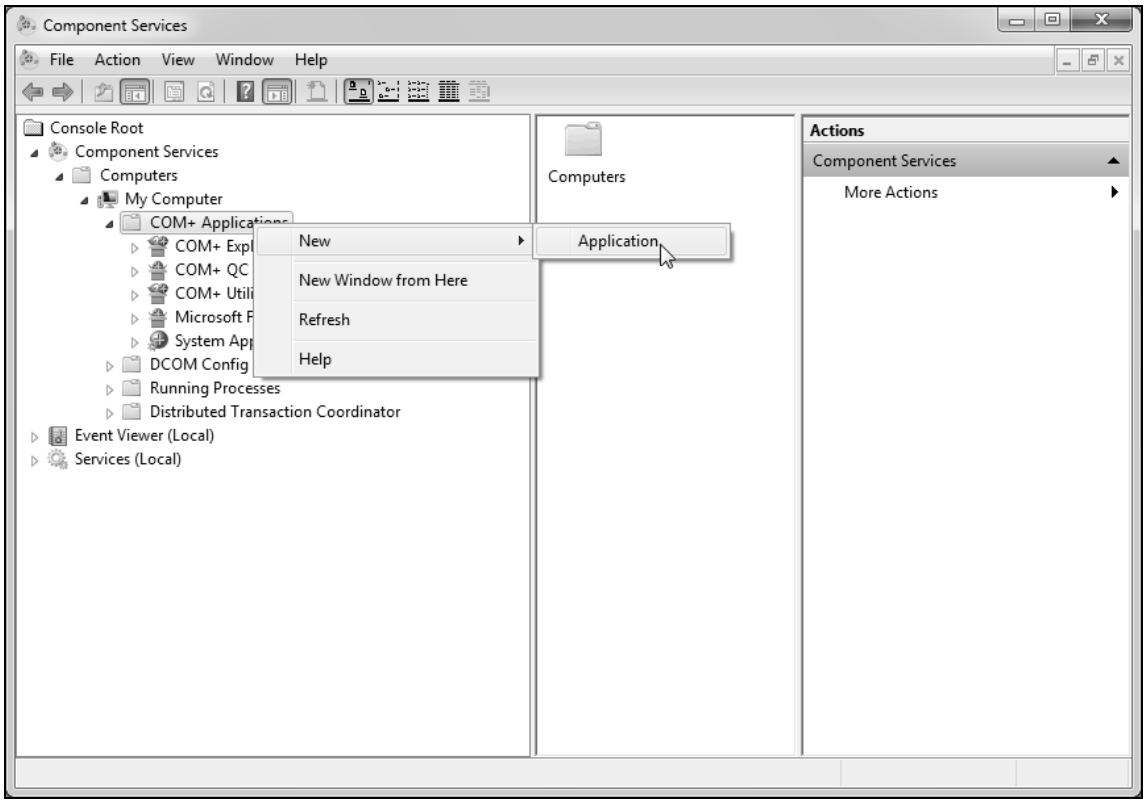

Figure A.12: Creating New Application in the CSAT

This will open the Welcome to the COM+ Application Install Wizard.

2. Click the 'Next' button to continue, as shown in Figure A.13:

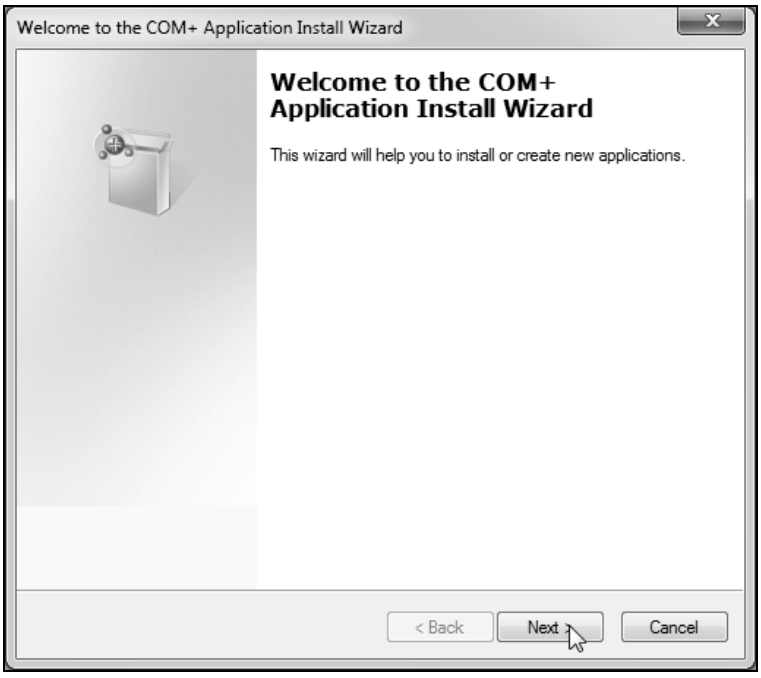

Figure A.13: Creating a New Application in the CSAT

3. Click the Create an empty application button to continue, as shown in Figure A.14:

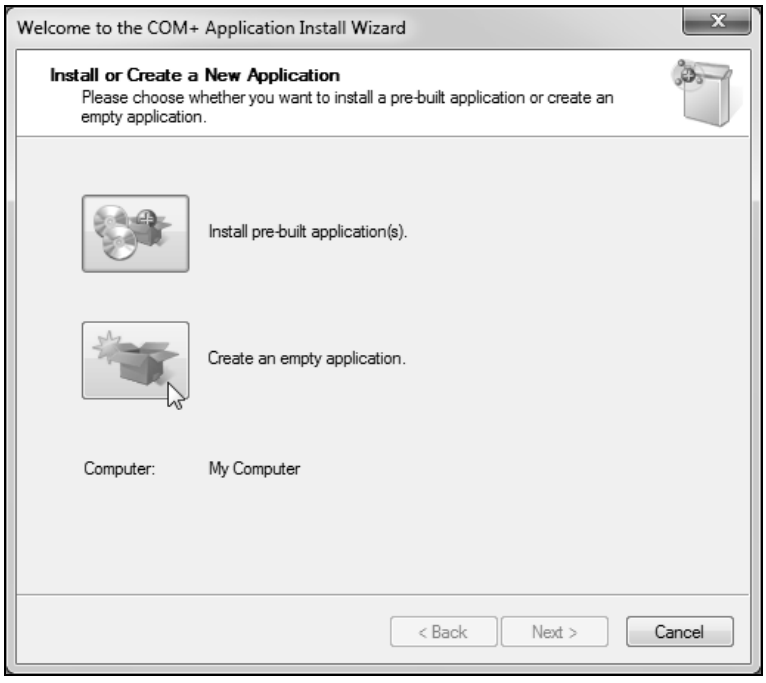

Figure A.14: Clicking the Create an Empty Application Button

4. In the next screen, type component as the application name and select Server application as Activation Type and then click the Next button to continue, as shown in Figure A.15:

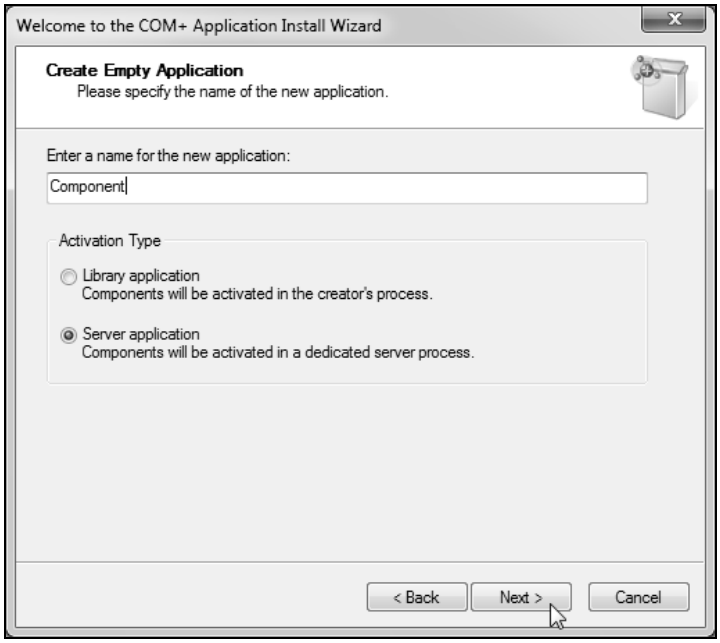

Figure A.15: Entering Application Name and Selecting Activation Type

5. In the next screen, select an application identity (under which account your application will run), and click the Next button to continue, as shown in Figure A.16:

Figure A.16: Selecting Application Identity

6. In the next screen, you can add different roles for your applications; in this case, accept the default settings and click the Next button to continue, as shown in Figure A.17:

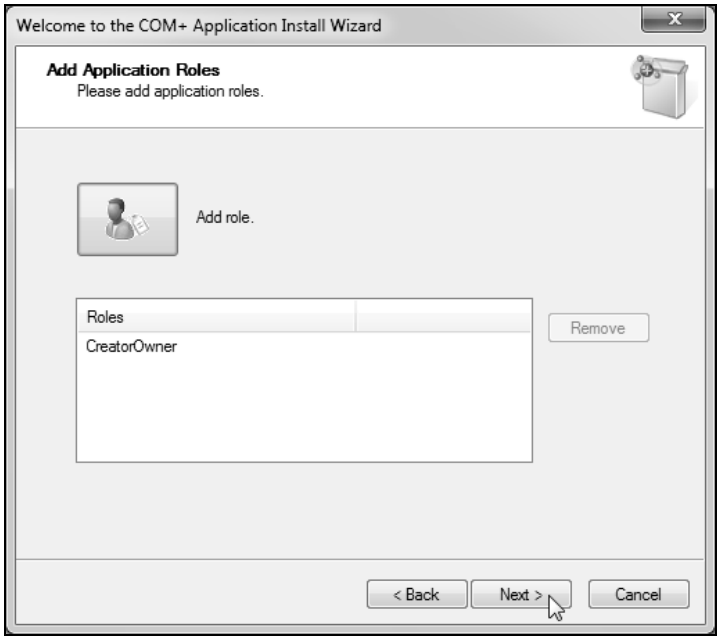

Figure A.17: Adding Roles for the Application

7. In the next screen, you can define different users for the roles you have added in the previous step. In this case, accept the default settings and click the Next button to continue, as shown in Figure A.18:

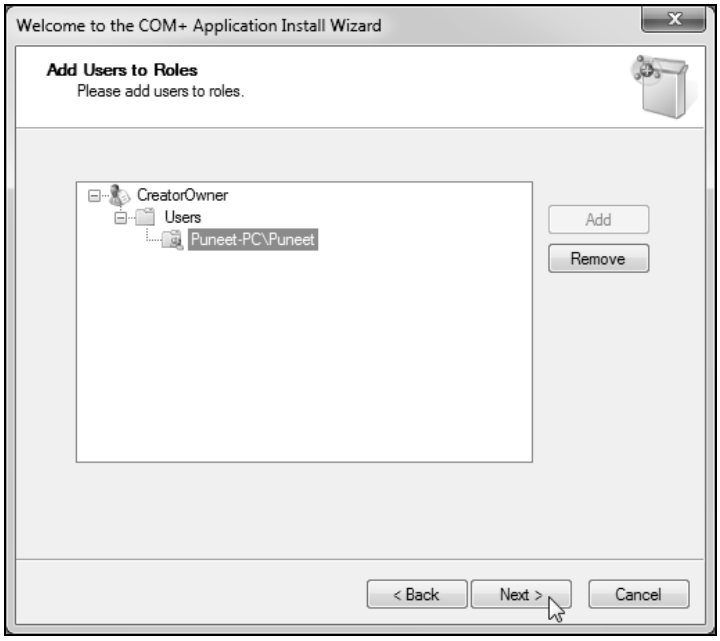

Figure A.18: Adding Users to the Roles

8. In the next screen, click the Finish button to close the wizard. This will add an empty application named component in the CSAT, as shown in Figure A.19:

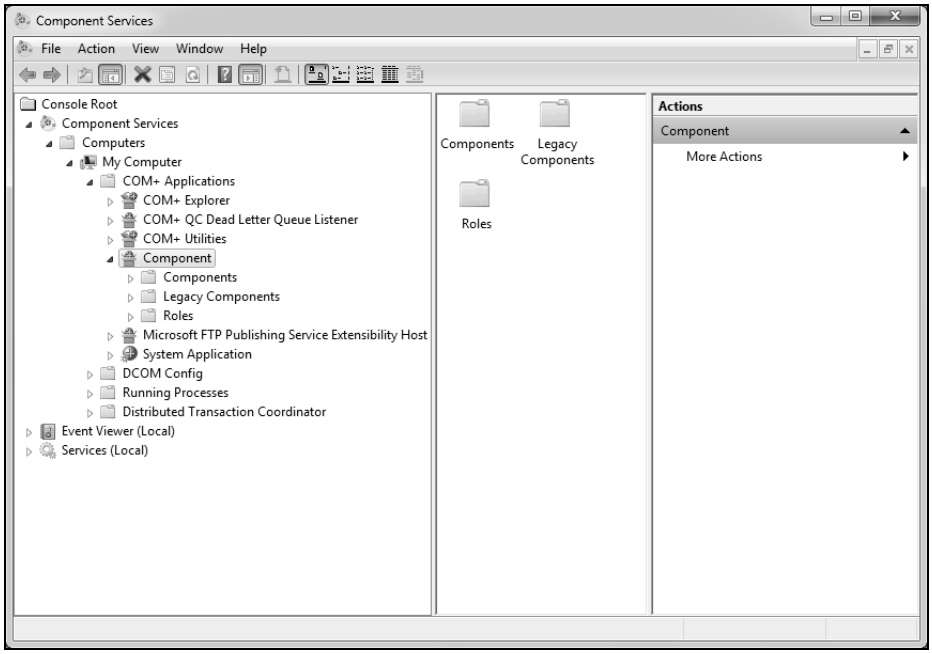

Figure A.19: Empty Application in the CSAT

9. In the next step, you now need to add component created earlier to this empty application. To do so, right-click the component node and select the New→Component option, as shown in Figure A.20:

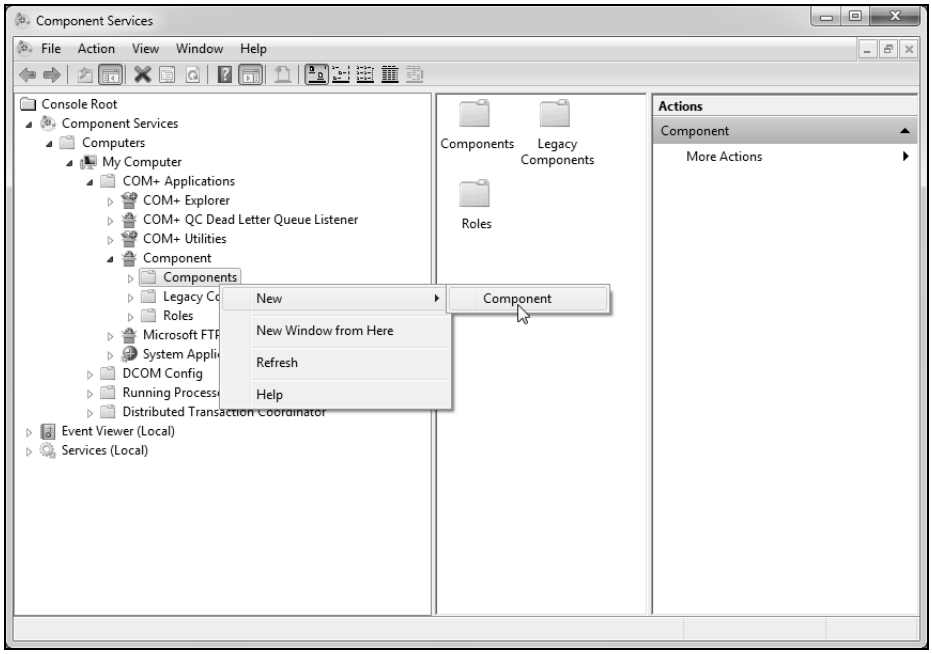

Figure A.20: Installing a Component in Application

10. As a consequence, the Welcome to the COM+ Component Install Wizard opens. Click the Next button to continue, as shown in Figure A.21:

Figure A.21: COM+ Component Install Wizard

11. In the next screen, click the Install new component(s) button to continue, as shown in Figure A.22:

Figure A.22: Clicking the Install New Component Button

12. This will open the Select files to install dialog box. Here, you need to select the `.dll` file, which is created by your application. Select the `.dll` file and click the Open button to continue, as shown in Figure A.23:

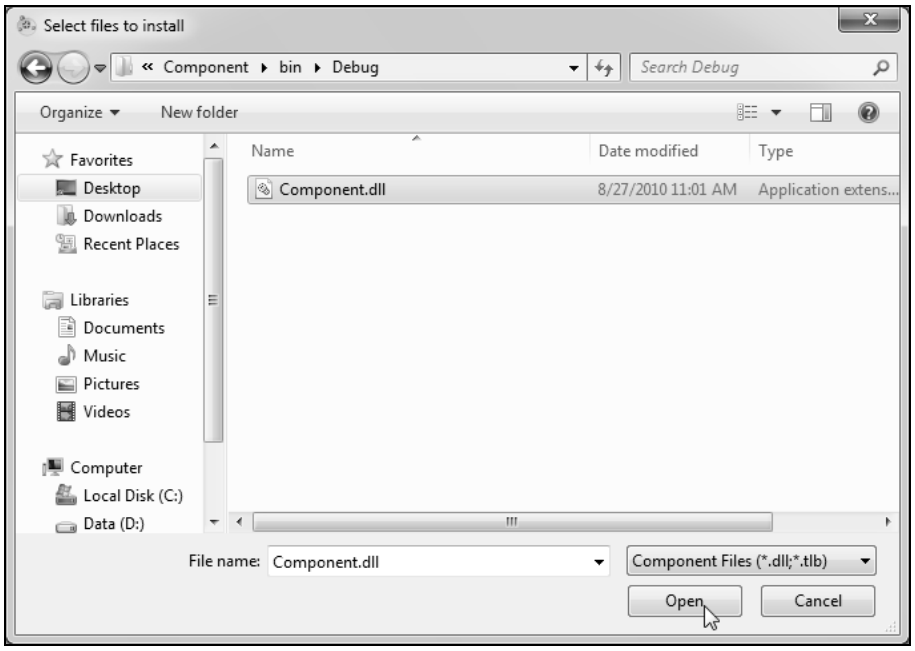

Figure A.23: Selecting the DLL File

13. In the next screen, your component will be listed and ready to install. Click the Next button to continue, as shown in Figure A.24:

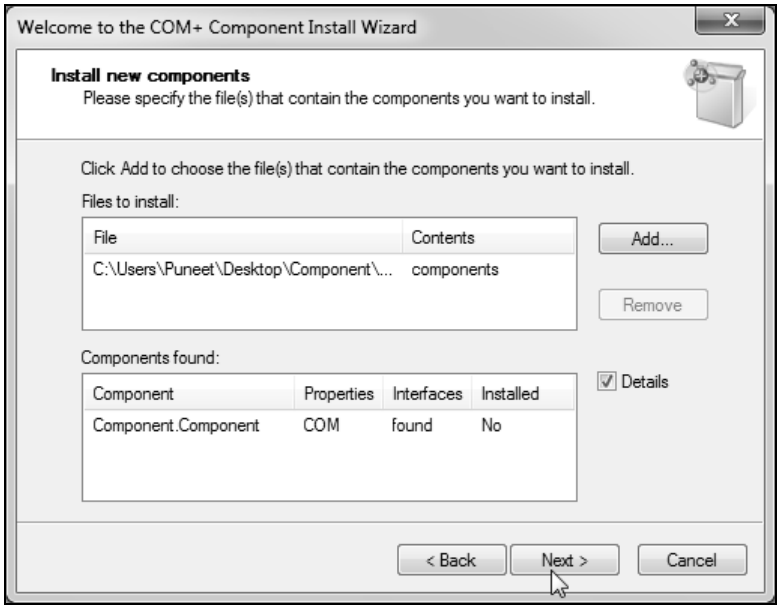

Figure A.24: Listing the Component to be Installed

14. Click the Finish button in the final screen of the wizard. This will add the component to the application with all its interfaces and methods, as shown in Figure A.25:

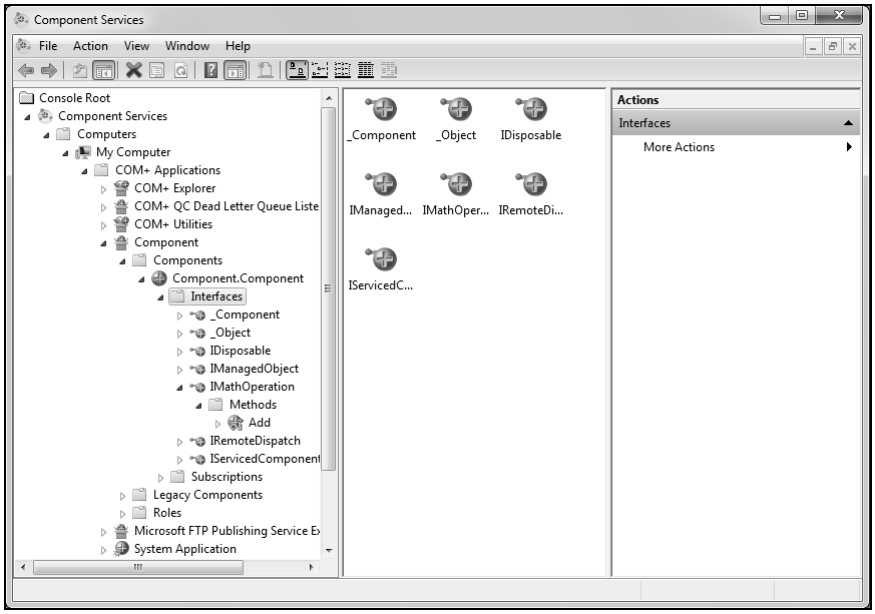

Figure A.25: Showing the Installed Components

This process of installing a component is lengthy and requires several steps. This can be a time-consuming process, if you need to install the component on several computers in your organization. To make it easy, you can now export the component as the MSI installer file using the CSAT, which will automate the process of component installation.

15. To export the component as the MSI installer file using the CSAT, right-click the application and select the Export option from the menu, as shown in Figure A.26:

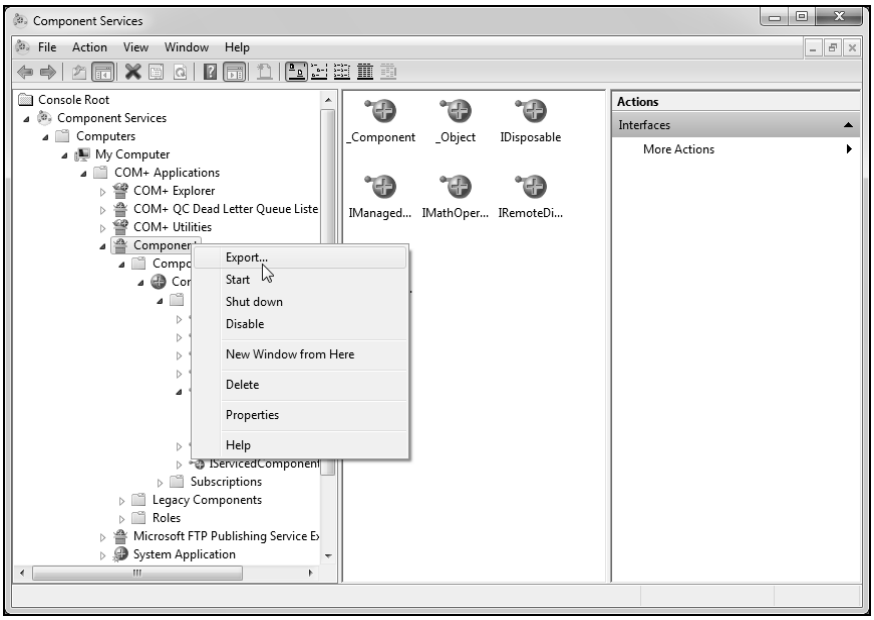

Figure A.26: Selecting the Export Option

This will open the COM+ Application Export Wizard.

16. Click the Next button to continue, as shown in Figure A.27:

Figure A.27: COM+ Application Export Wizard

17. In the next screen, type the full path and name you want to give to your MSI file, and also specify the type of an application; in this case, select Server application and click the Next button to continue, as shown in Figure A.28:

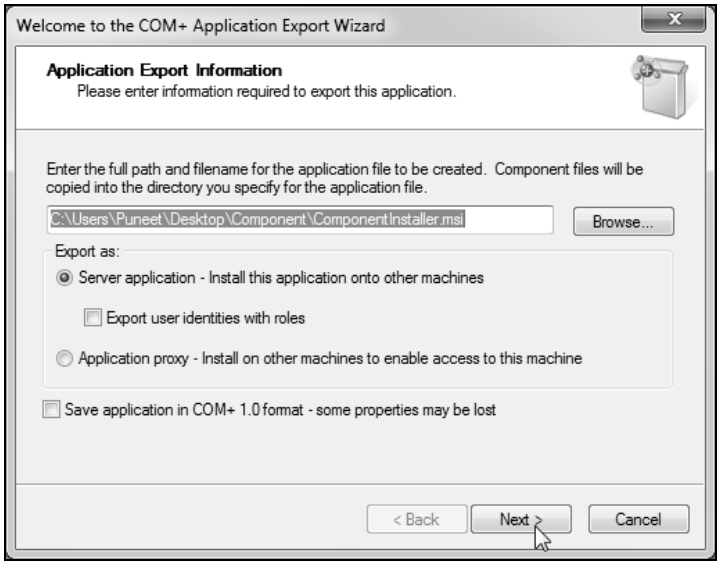

Figure A.28: Specifying File Name and Application Type

This will generate the MSI installer and takes you to the final screen of the wizard.

18. Click the Finish button to close the wizard. Let's install this component 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 then double-click the MSI file to launch the installer, as shown in Figure A.29:

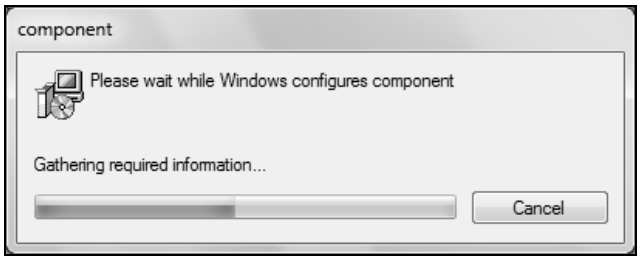

Figure A.29: Installing Component

This will install the COM+ application on your computer. You can check this by opening the CSAT. The installed application will be listed in the COM+ Applications node, as shown in Figure A.30:

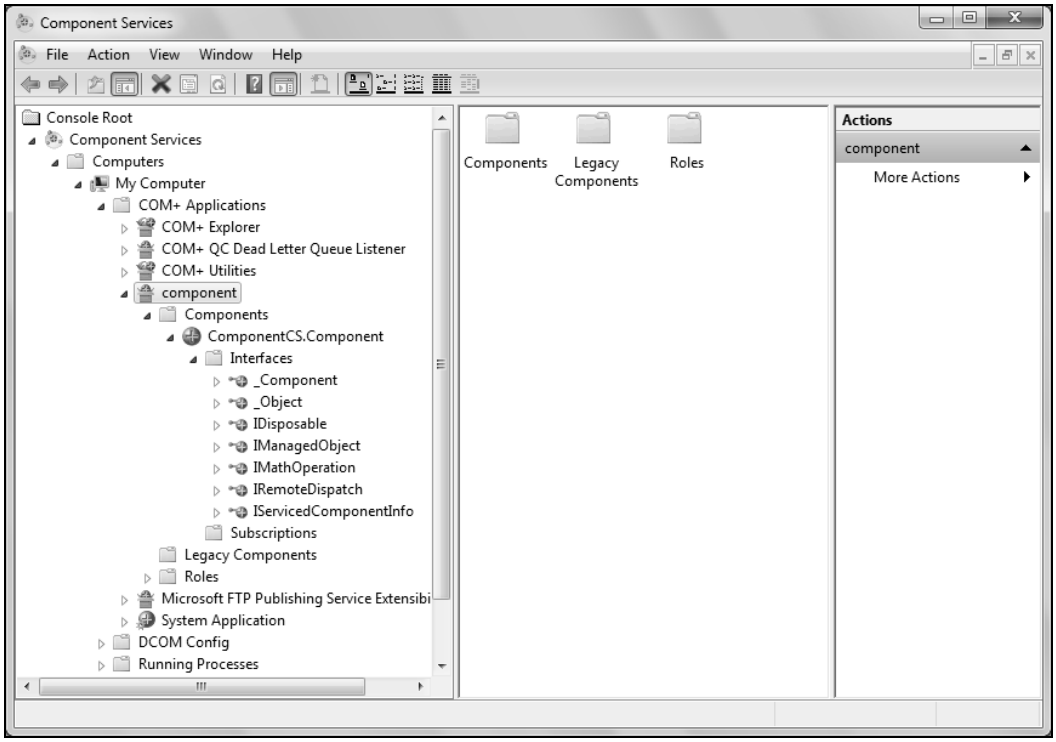

Figure A.30: CSAT Showing the Installed Application

Exposing COM+ Applications as WCF Service

Now that you have a component created and installed as a COM+ application, you can use this component with the COM or COM+ techniques, but it only works fine if a connection is created. Thus, to make it available to users with Internet connection, .NET 4.0 Framework has provided with the provision of exposing COM+ applications as a Windows Communication Foundation (WCF) service. Perform the following steps to publish a component:

1. Create a folder named component on your computer and then create a new virtual directory in the Internet Information Server (IIS) and point it to the directory you have just created. In the next step, open the Microsoft Service Configuration Editor as administrator by selecting C:→ Program Files→ Microsoft SDKs → Windows → v7.0A → bin → NETFX 4.0 Tools → SvcConfigEditor, as shown in Figure A.31:

Figure A.31: Service Configuration Editor

2. Integrate a new COM+ application as a WCF service by selecting the File→Integrate→COM+ Application option, as shown in Figure A.32:

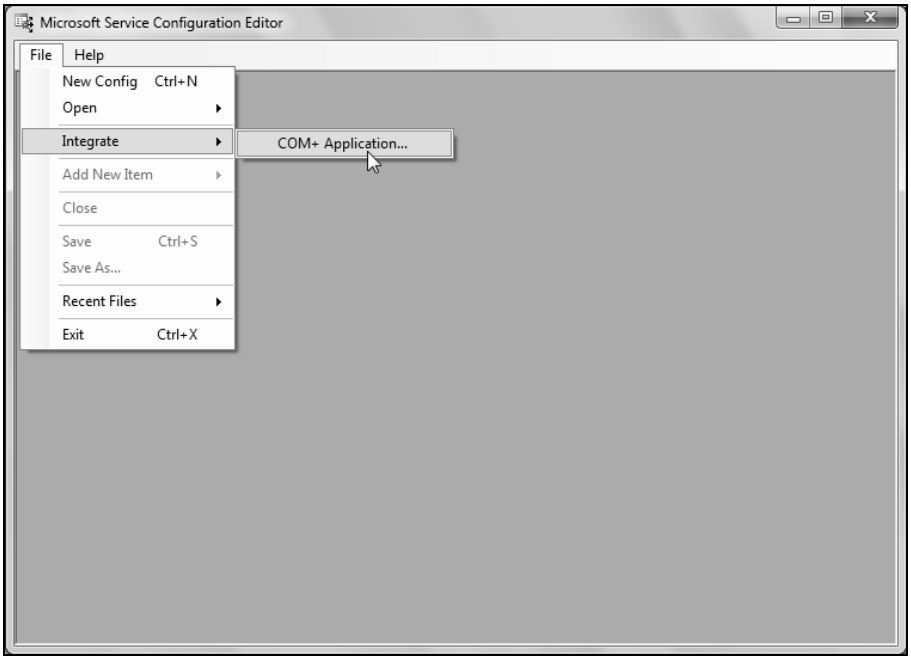

Figure A.32: Starting COM+ Integration Wizard

3. This will start the COM+ Integration Wizard. In the first screen of the wizard, you need to select the interface of the component that you are going to publish and then click the Next button to continue, as shown in Figure A.33:

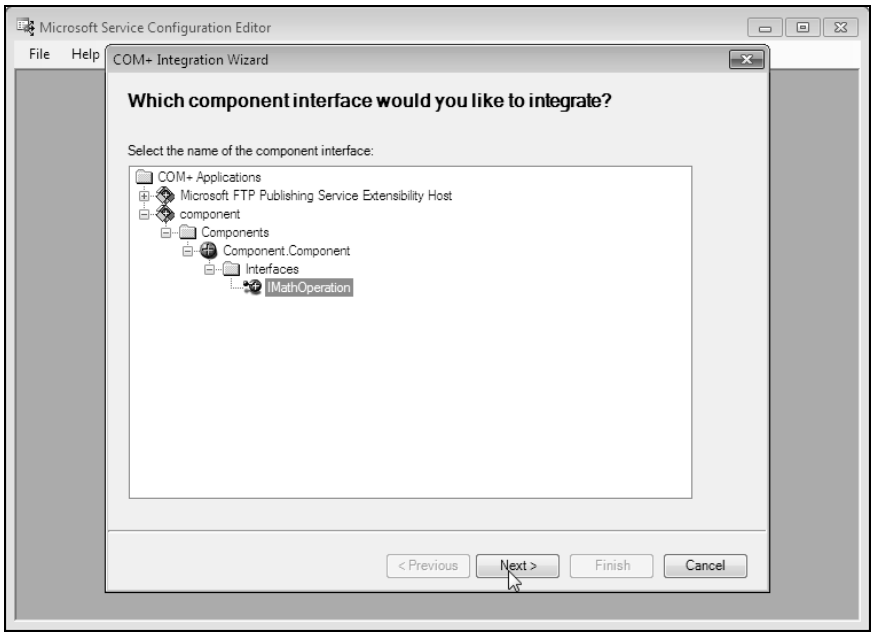

Figure A.33: Selecting the Interface

4. In the next step, select the method that needs to be integrated and click the Next button to continue, as shown in Figure A.34:

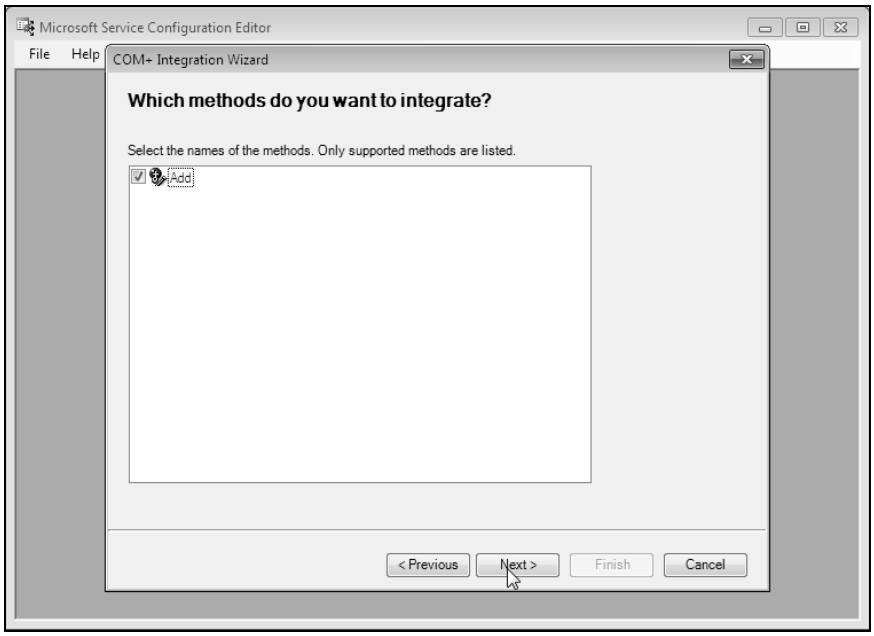

Figure A.34: Selecting the Method

5. In the next screen, you need to select the hosting mode. In this case, select Web hosted and then click the Next button to continue, as shown in Figure A.35:

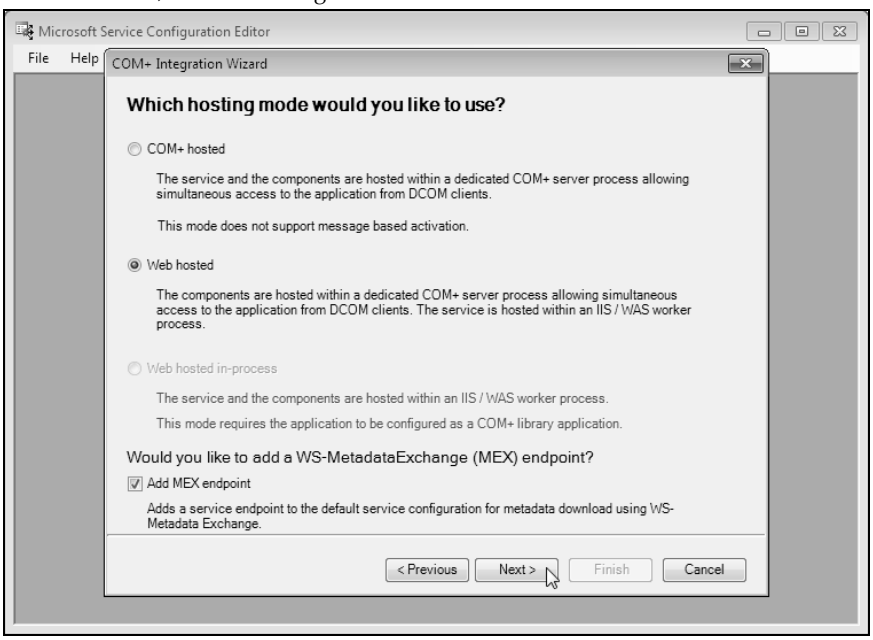

Figure A.35: Selecting Hosting Mode

6. In the next screen, select the virtual directory you have created in earlier steps and click the Next button to continue, as shown in Figure A.36:

Figure A.36: Selecting Virtual Directory

7. The next screen displays the information about the service configuration ready to be created. Click the Next button to continue, as shown in Figure A.37:

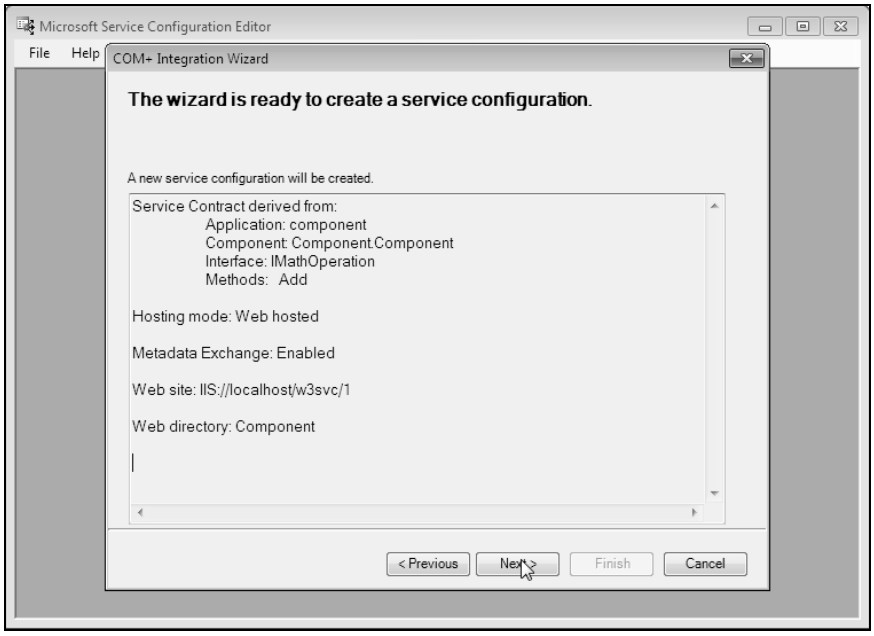

Figure A.37: Service Configuration Information Displayed by Wizard

As a result the wizard now publishes the WCF service and shows the final screen of the wizard displaying information about the operation performed.

8. Click the Finish button to close the wizard, as shown in Figure A.38:

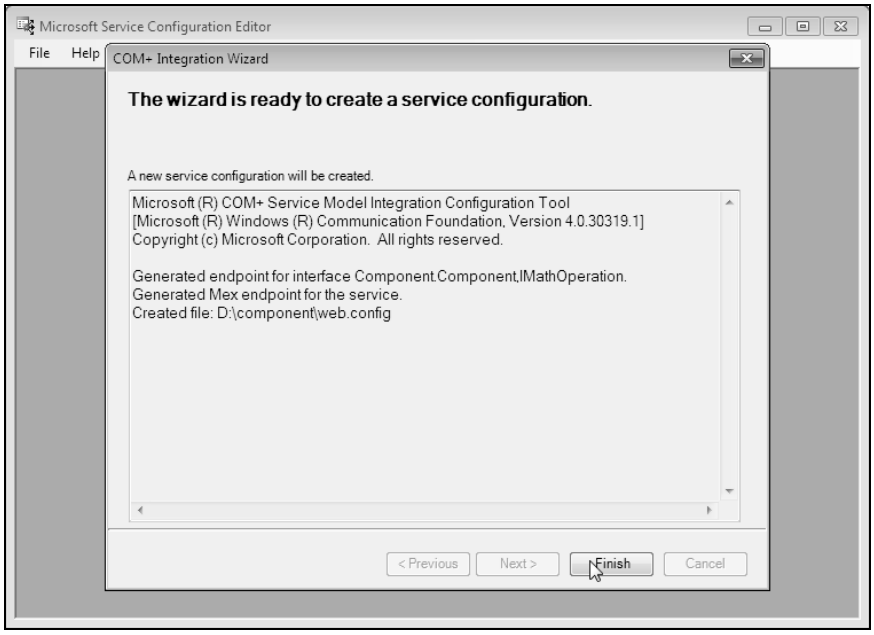

Figure A.38: Closing the Wizard

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 is shown in Figure A.39:

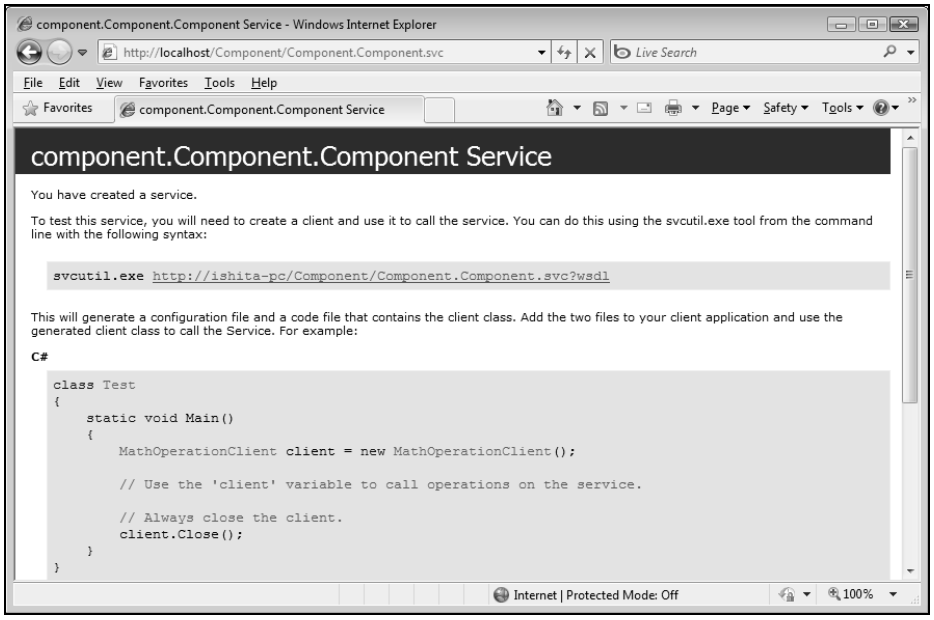

Figure A.39: Testing the WCF Service in Browser

9. Click the link appearing on the page. The Web Services Description Language (WSDL) for the WCF service is displayed, as shown in Figure A.40:

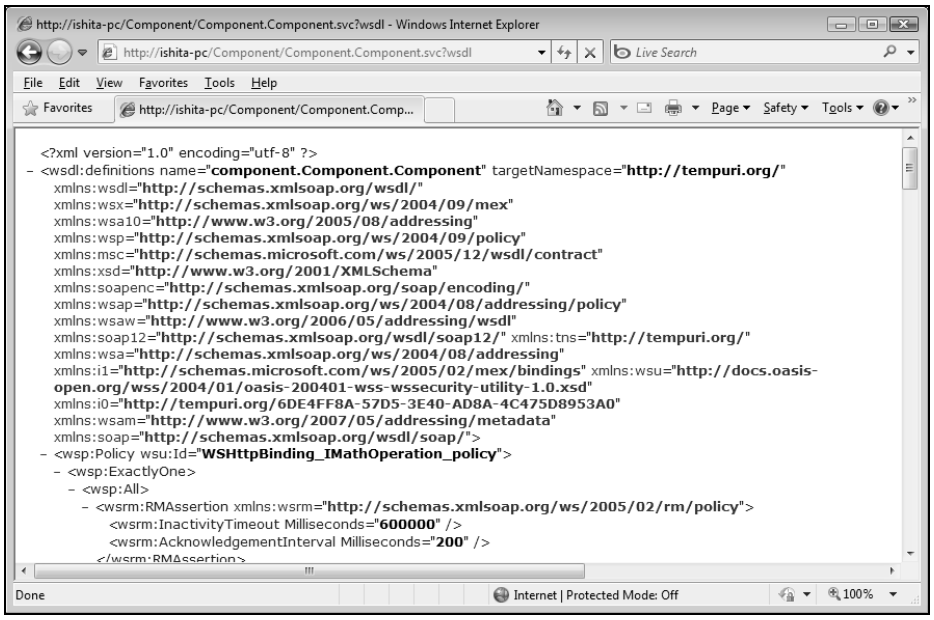

Figure A.40: WSDL Displaying a Single Link for the Web Service

You can find the code for the WSDL file in Listing A.3:

Listing A.3: Showing the Code for 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/Always
oRecipient">
- <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/Always
oRecipient">
- <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: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: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://ishita-pc/Component/Component.Component.svc?wsdl=wsdl0" />
      <wsdl:import namespace="http://schemas.microsoft.com/ws/2005/02/mex/bindings"
      location="http://ishita-pc/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://ishita-
       pc/Component/Component.Component.svc/IMathOperation" />
    - <wsa10:EndpointReference>
      <wsa10:Address>http://ishita-
       pc/Component/Component.Component.svc/IMathOperation</wsa10:Address>
    - <Identity xmlns="http://schemas.xmlsoap.org/ws/2006/02/addressingidentity">
      <Spn>host/Ishita-PC</Spn>
      </Identity>
      </wsa10:EndpointReference>
      </wsdl:port>
    - <wsdl:port name="MetadataExchangeHttpBinding_IMetadataExchange"
       binding="i1:MetadataExchangeHttpBinding_IMetadataExchange">
      <soap12:address location="http://ishita-pc/Component/Component.Component.svc/mex" />
    - <wsa10:EndpointReference>
      <wsa10:Address>http://ishita-pc/Component/Component.Component.svc/mex</wsa10:Address>
      </wsa10:EndpointReference>
      </wsdl:port>
      </wsdl:service>
      </wsdl:definitions>
```

Creating Services Without Components

To create a service without components, let's create a new Web application named `ServicesWithoutComponents` (can be found on the CD-ROM as `ServicesWithoutComponentsVB` and `ServicesWithoutComponentsCS`). Perform the following steps:

1. Create a database on your computer named Products with two tables named Products and ProductsType. Table A.6 displays the schema for the Products table:

Table A.6: Schema for Products Table

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

NOTE

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

Table A.7 displays the schema for the ProductsType table:

Table A.7: Schema for ProductsType Table

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

2. Add three classes named `Products`, `ProductType`, and `TransactionClass`.

In the `Products` and `ProductType` classes, add methods to add new products and product types. In the `TransactionClass` class, add a method that calls the methods present in the `Products` and `ProductType` classes to add new products and product types. Both the operations need to be performed simultaneously. Hence, you need to use the transaction feature of the `ServiceConfig` object in the `TransactionClass` class.

3. Add the code, shown in Listing A.4, in the `Products` class file.

Listing A.4: Showing the Code for the `Products` Class File

In VB

```
Imports System.Data
Imports System.Data.SqlClient
Public Class Products
  Public Sub New()
```

```
    End Sub
    Public Function AddProducts(ByVal name As String, ByVal description As String) As
      Integer
        Dim productID As Integer
        Dim connString As String = "Data Source=UMAR-PC\SQLEXPRESS;Initial
          Catalog=Products;Integrated Security=True"
        Dim commandName As String = "AddProducts"
        Using conn As New SqlConnection(connString)
                conn.Open()
                Dim command As New SqlCommand(commandName, conn)
                command.CommandType = CommandType.StoredProcedure
                Dim paramName As New SqlParameter("@Name", SqlDbType.VarChar, 50)
                paramName.Direction = ParameterDirection.Input
                paramName.Value = name
                command.Parameters.Add(paramName)
                Dim paramDesc As New SqlParameter("@Description", SqlDbType.VarChar, 250)
                paramDesc.Direction = ParameterDirection.Input
                paramDesc.Value = description
                command.Parameters.Add(paramDesc)
                Dim paramReturnValue As New SqlParameter("@@identity", SqlDbType.VarChar, 250)
                paramReturnValue.Direction = ParameterDirection.ReturnValue
                command.Parameters.Add(paramReturnValue)
                command.ExecuteNonQuery()
                productID = CInt(Fix(command.Parameters("@@identity").Value))
                Console.WriteLine(productID)
                Console.ReadLine()
        End Using
        Return productID
    End Function
End Class
```

In C#

```
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 Products()
        {
        }
        public int AddProducts(string name, string description)
        {
                int productID;
                string connString = @"Data Source=UMAR-PC\SQLEXPRESS;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;
                        Console.WriteLine(productID);
                        Console.ReadLine();
                }
                return productID;
        }
    } }
```

NOTE

Change the connection string according to your system setting.

In the preceding listing, the `Products` class contains a method named `AddProducts`, which uses a stored procedure named `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.

4. Now, open the SQL Server 2008 and create the `AddProducts` procedure as followings:

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

5. Add the code, shown in Listing A.5, to create the `ProductType` class.

Listing A.5: Showing the Code for the `ProductsType` Class File

In VB

```
Imports System.Data
Imports System.Data.SqlClient
Public Class ProductType
   Public Sub New()
   End Sub
   Public Function AddProductType(ByVal productID As Integer, ByVal productType As String)
    As Integer
         Dim productTypeID As Integer
         Dim connString As String = "Data Source=UMAR-PC\SQLEXPRESS;Initial
         Catalog=Products;Integrated Security=True"
         Dim commandName As String = "AddProductTypes"
         Using conn As New SqlConnection(connString)
                 conn.Open()
                 Dim command As New SqlCommand(commandName, conn)
                 command.CommandType = CommandType.StoredProcedure
                 Dim paramProductID As New SqlParameter("@ProductID", SqlDbType.Int)
                 paramProductID.Direction = ParameterDirection.Input
                 paramProductID.Value = productID
                 command.Parameters.Add(paramProductID)
                 Dim paramProductType As New SqlParameter("@ProductType",
                   SqlDbType.VarChar, 50)
                 paramProductType.Direction = ParameterDirection.Input
                 paramProductType.Value = productType
                 command.Parameters.Add(paramProductType)
                 Dim paramReturnValue As New SqlParameter("@@identity", SqlDbType.VarChar,
                   250)
                 paramReturnValue.Direction = ParameterDirection.ReturnValue
                 command.Parameters.Add(paramReturnValue)
                 command.ExecuteNonQuery()
                 productTypeID = CInt(Fix(command.Parameters("@@identity").Value))
         End Using
         Return productTypeID
   End Function
End Class
```

In C#

```
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 ProductType()
        {
        }
        public int AddProductType(int productID, string productType)
        {
                int productTypeID;
                string connString = @"Data Source=UMAR-PC\SQLEXPRESS;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;
        }
  }
}
```

In the preceding listing, 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.

6. Create the `AddProductTypes` stored procedure in SQL Server 2008.

Here is 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
```

7. Add the code for the `TransactionClass` class, which ensures that the details related to the products and product types are added as a single transaction. You can find the code for the `TransactionClass` file in Listing A.6:

Listing A.6: Showing the Code for the `TransactionClass` File

In VB

```
Imports System.Data
Imports System.Data.SqlClient
Imports System.EnterpriseServices
Public Class TransactionClass
  Public Sub New()
  End Sub
  Public Function AddDetails(ByVal prodName As String, ByVal prodDescription As String,
    ByVal prodType As String) As String
        Dim config As New ServiceConfig()
        config.Transaction = TransactionOption.Required
        ServiceDomain.Enter(config)
        Try
                Dim prod As New Products()
                Dim ProdID As Integer = prod.AddProducts(prodName, prodDescription)
                Dim pType As New ProductType()
                pType.AddProductType(ProdID, prodType)
                Catch
                Throw
                Finally
                ServiceDomain.Leave()
        End Try
        Return "Data entered successfully"
  End Function
End Class
```

In C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.EnterpriseServices;
namespace TransactionComponent
{
  public class TransactionClass
  {
        public 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 listing, an object of the `ServiceConfig` class is created, which starts the `AddProducts` method the moment 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 created. Since a transaction is used to maintain consistency of data, the statements between the `Enter` and the `Leave` methods will ensure that either the transaction will commit successfully, or if any exception arises during the execution, it will rollback.

8. In the `ServicesWithoutComponents` project, add a `Button` control to the form and add the following code snippet for the `Click` event of the button:

```
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"));
}
```

In the preceding code, the `AddDetails` method of the `TransactionClass` class returns a string value if the method executes successfully. In case any exception arises during the execution, the details are not added to the database. Now, add the code in the code-behind file of the `Default.aspx` page as shown in Listing A.7:

Listing A.7: Showing the code for the Code-Behind File of the `Default.aspx` Page

In VB

```
Imports TransactionComponent
Partial Class _Default
  Inherits System.Web.UI.Page
  Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Handles Button1.Click
       Dim trans As New TransactionClass()
       Label1.Text = trans.AddDetails("Table Lamp", "A table lamp with the picture of
        flowers", "Show piece")
  End Sub
End Class
```

In C#

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using TransactionComponent;
using System.Web.UI.WebControls;
using System.Text;
namespace ServicesWithoutComponents
{
  public partial class _Default : System.Web.UI.Page
  {
        protected void Button1_Click(object sender, EventArgs e)
        {
              TransactionClass trans = new TransactionClass();
              Label1.Text=trans.AddDetails("Table Lamp", "A table lamp with the picture
                of flowers", "Show piece");
        }
  }
}
```

9. Now, set the `ServicesWithoutComponents` project as the startup project and the `Default.aspx` page as the startup form.
10. Press the F5 key to execute the application and click the Button control to see the result, as shown in Figure A.41:

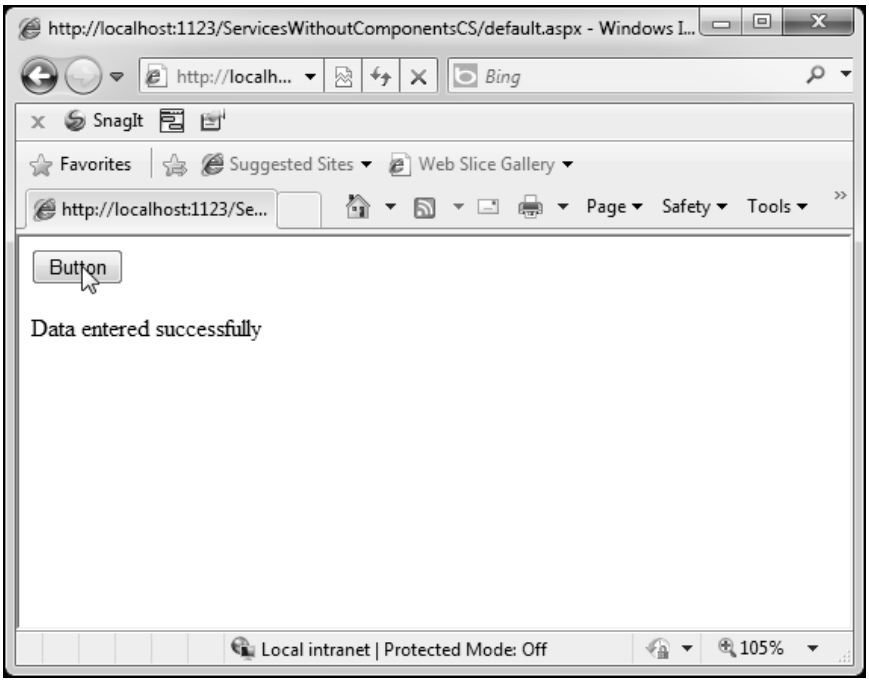

Figure A.41: Using the ServiceDomain and ServiceConfig Classes