23

Web Services

A client is to me a mere unit, a factor in a problem.
—Sir Arthur Conan Doyle

...if the simplest things of nature have a message that you understand, rejoice, for your soul is alive.
—Eleonora Duse

Objectives

In this chapter you'll learn:

- How to create WCF web services.
- How XML, JSON, XML-Based Simple Object Access Protocol (SOAP) and Representational State Transfer Architecture (REST) enable WCF web services.
- The elements that comprise WCF web services, such as service references, service endpoints, service contracts and service bindings.
- How to create a client that consumes a WCF web service.
- How to use WCF web services with Windows and web applications.
- How to use session tracking in WCF web services to maintain state information for the client.
- How to pass user-defined types to a WCF web service.

Outline

23.1 Introduction

This chapter introduces **Windows Communication Foundation (WCF)** services. WCF is a set of technologies for building distributed systems in which system components communicate with one another over networks. In earlier versions of .NET, the various types of communication used different technologies and programming models. WCF uses a common framework for all communication between systems, so you need to learn only one programming model to use WCF.

This chapter focuses on WCF web services, which promote software reusability in distributed systems that typically execute across the Internet. A **web service** is a class that allows its methods to be called by methods on other machines via common data formats and protocols, such as XML (see Chapter 21), JSON (Section 23.5) and HTTP. In .NET, the over-the-network method calls are commonly implemented through **Simple Object Access Protocol (SOAP)** or the **Representational State Transfer (REST)** architecture. SOAP is an XML-based protocol describing how to mark up requests and responses so that they can be sent via protocols such as HTTP. SOAP uses a standardized XML-based format to enclose data in a message that can be sent between a client and a server. REST

is a network architecture that uses the web's traditional request/response mechanisms such as `GET` and `POST` requests. REST-based systems do not require data to be wrapped in a special message format.

We build the WCF web services presented in this chapter in Visual Web Developer 2010 Express, and we create client applications that invoke these services using both Visual Basic 2010 Express and Visual Web Developer 2010 Express. Full versions of Visual Studio 2010 include the functionality of both Express editions.

Requests to and responses from web services created with Visual Web Developer are typically transmitted via SOAP or REST, so any client capable of generating and processing SOAP or REST messages can interact with a web service, regardless of the language in which the web service is written. We say more about SOAP and REST in Section 23.3 and Section 23.4, respectively.

23.2 WCF Services Basics

Microsoft's Windows Communication Foundation (WCF) was created as a single platform to encompass many existing communication technologies. WCF increases productivity, because you learn only one straightforward programming model. Each WCF service has three key components—addresses, bindings and contracts (usually called the ABCs of a WCF service):

- An **address** represents the service's location (also known as its **endpoint**), which includes the protocol (for example, HTTP) and network address (for example, `www.deitel.com`) used to access the service.
- A **binding** specifies how a client communicates with the service (for example, SOAP, REST, and so on). Bindings can also specify other options, such as security constraints.
- A **contract** is an interface representing the service's methods and their return types. The service's contract allows clients to interact with the service.

The machine on which the web service resides is referred to as a **web service host**. The client application that accesses the web service sends a method call over a network to the web service host, which processes the call and returns a response over the network to the application. This kind of distributed computing benefits systems in various ways. For example, an application without direct access to data on another system might be able to retrieve this data via a web service. Similarly, an application lacking the processing power necessary to perform specific computations could use a web service to take advantage of another system's superior resources.

23.3 Simple Object Access Protocol (SOAP)

The Simple Object Access Protocol (SOAP) is a platform-independent protocol that uses XML to make remote procedure calls, typically over HTTP. Each request and response is packaged in a **SOAP message**—an XML message containing the information that a web service requires to process the message. SOAP messages are written in XML so that they're computer readable, human readable and platform independent. Most **firewalls**—security barriers that restrict communication among networks—allow HTTP traffic to pass through, so that clients can browse the Internet by sending requests to and receiving re-

sponses from web servers. Thus, SOAP-based services can send and receive SOAP messages over HTTP connections with few limitations.

SOAP supports an extensive set of types. The **wire format** used to transmit requests and responses must support all types passed between the applications. SOAP types include the primitive types (for example, `Integer`), as well as `DateTime`, `XmlNode` and others. SOAP can also transmit arrays of these types. In Section 23.11, you'll see that you can also transmit user-defined types in SOAP messages.

When a program invokes a method of a SOAP web service, the request and all relevant information are packaged in a SOAP message enclosed in a **SOAP envelope** and sent to the server on which the web service resides. When the web service receives this SOAP message, it parses the XML representing the message, then processes the message's contents. The message specifies the method that the client wishes to execute and the arguments the client passed to that method. Next, the web service calls the method with the specified arguments (if any) and sends the response back to the client in another SOAP message. The client parses the response to retrieve the method's result. In Section 23.6, you'll build and consume a basic SOAP web service.

23.4 Representational State Transfer (REST)

Representational State Transfer (REST) refers to an architectural style for implementing web services. Such web services are often called **RESTful web services**. Though REST itself is not a standard, RESTful web services are implemented using web standards. Each operation in a RESTful web service is identified by a unique URL. Thus, when the server receives a request, it immediately knows what operation to perform. Such web services can be used in a program or directly from a web browser. The results of a particular operation may be cached locally by the browser when the service is invoked with a `GET` request. This can make subsequent requests for the same operation faster by loading the result directly from the browser's cache. Amazon's web services (`aws.amazon.com`) are RESTful, as are many others.

RESTful web services are alternatives to those implemented with SOAP. Unlike SOAP-based web services, the request and response of REST services are not wrapped in envelopes. REST is also not limited to returning data in XML format. It can use a variety of formats, such as XML, JSON, HTML, plain text and media files. In Sections 23.7–23.8, you'll build and consume basic RESTful web services.

23.5 JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) is an alternative to XML for representing data. JSON is a text-based data-interchange format used to represent objects in JavaScript as collections of name/value pairs represented as `Strings`. It is commonly used in Ajax applications. JSON is a simple format that makes objects easy to read, create and parse, and allows programs to transmit data efficiently across the Internet because it is much less verbose than XML. Each JSON object is represented as a list of property names and values contained in curly braces, in the following format:

```
{ propertyName1 : value1, propertyName2 : value2 }
```

Arrays are represented in JSON with square brackets in the following format:

```
[ value1, value2, value3 ]
```

Each value in an array can be a string, a number, a JSON object, `true`, `false` or `null`. To appreciate the simplicity of JSON data, examine this representation of an array of address-book entries

```
[ { first: 'Cheryl', last: 'Black' },
  { first: 'James', last: 'Blue' },
  { first: 'Mike', last: 'Brown' },
  { first: 'Meg', last: 'Gold' } ]
```

Many programming languages now support the JSON data format.

23.6 Publishing and Consuming SOAP-Based WCF Web Services

This section presents our first example of **publishing** (enabling for client access) and **consuming** (using) a web service. We begin with a SOAP-based web service.

23.6.1 Creating a WCF Web Service

To build a SOAP-based WCF web service in Visual Web Developer, you first create a project of type **WCF Service**. SOAP is the default protocol for WCF web services, so no special configuration is required to create them. Visual Web Developer then generates files for the WCF service code, an **SVC file** (`Service.svc`, which provides access to the service), and a **`Web.config`** file (which specifies the service's binding and behavior).

Visual Web Developer also generates code files for the **WCF service class** and any other code that is part of the WCF service implementation. In the service class, you define the methods that your WCF web service makes available to client applications.

23.6.2 Code for the `WelcomeSOAPXMLService`

Figures 23.1 and 23.2 present the code-behind files for the `WelcomeSOAPXMLService` WCF web service that you build in Section 23.6.3. When creating services in Visual Web Developer, you work almost exclusively in the code-behind files. The service provides a method that takes a name (represented as a `String`) as an argument and appends it to the welcome message that is returned to the client. We use a parameter in the method definition to demonstrate that a client can send data to a web service.

Figure 23.1 is the service's interface, which describes the service's contract—the set of methods and properties the client uses to access the service. The **`ServiceContract`** attribute (line 4) exposes a class that implements this interface as a WCF web service. The **`OperationContract`** attribute (line 7) exposes the `Welcome` method to clients for remote calls. Optional parameters can be assigned to these contracts to change the data format and method behavior, as we'll show in later examples.

```
1  ' Fig. 23.1: IWelcomeSOAPXMLService.vb
2  ' WCF web service interface that returns a welcome message through SOAP
3  ' protocol and XML format.
4  <ServiceContract()>
```

Fig. 23.1 | WCF web service interface that returns a welcome message through SOAP protocol and XML format. (Part 1 of 2.)

```
5  Public Interface IWelcomeSOAPXMLService
6     ' returns a welcome message
7     <OperationContract()>
8     Function Welcome(ByVal yourName As String) As String
9  End Interface ' IWelcomeSOAPXMLService
```

Fig. 23.1 | WCF web service interface that returns a welcome message through SOAP protocol and XML format. (Part 2 of 2.)

Figure 23.2 defines the class that implements the interface declared as the `ServiceContract`. Lines 8–13 define the method `Welcome`, which returns a `String` welcoming you to WCF web services. Next, we build the web service from scratch.

```
1  ' Fig. 23.2: WelcomeSOAPXMLService.vb
2  ' WCF web service that returns a welcome message through SOAP protocol and
3  ' XML format.
4  Public Class WelcomeSOAPXMLService
5     Implements IWelcomeSOAPXMLService
6
7     ' returns a welcome message
8     Public Function Welcome(ByVal yourName As String) As String _
9        Implements IWelcomeSOAPXMLService.Welcome
10
11       Return "Welcome to WCF Web Services with SOAP and XML, " &
12          yourName & "!"
13    End Function ' Welcome
14 End Class ' WelcomeSOAPXMLService
```

Fig. 23.2 | WCF web service that returns a welcome message through the SOAP protocol and XML format.

23.6.3 Building a SOAP WCF Web Service

In the following steps, you create a **WCF Service** project for the `WelcomeSOAPXMLService` and test it using the built-in ASP.NET Development Server that comes with Visual Web Developer Express and Visual Studio.

Step 1: Creating the Project

To create a project of type **WCF Service**, select **File > New Web Site...** to display the **New Web Site** dialog (Fig. 23.3). Select the **WCF Service** template. Select **File System** from the **Location** drop-down list to indicate that the files should be placed on your local hard disk. By default, Visual Web Developer places files on the local machine in a directory named `WCFService1`. Rename this folder to `WelcomeSOAPXMLService`. We modified the default path as well. Click **OK** to create the project.

Step 2: Examining the Newly Created Project

After you create the project, the code-behind file `Service.vb`, which contains code for a simple web service, is displayed by default. If the code-behind file is not open, open it by double clicking the file in the **App_Code** directory listed in the **Solution Explorer**. By default, a new code-behind file implements an interface named `IService` that is marked with

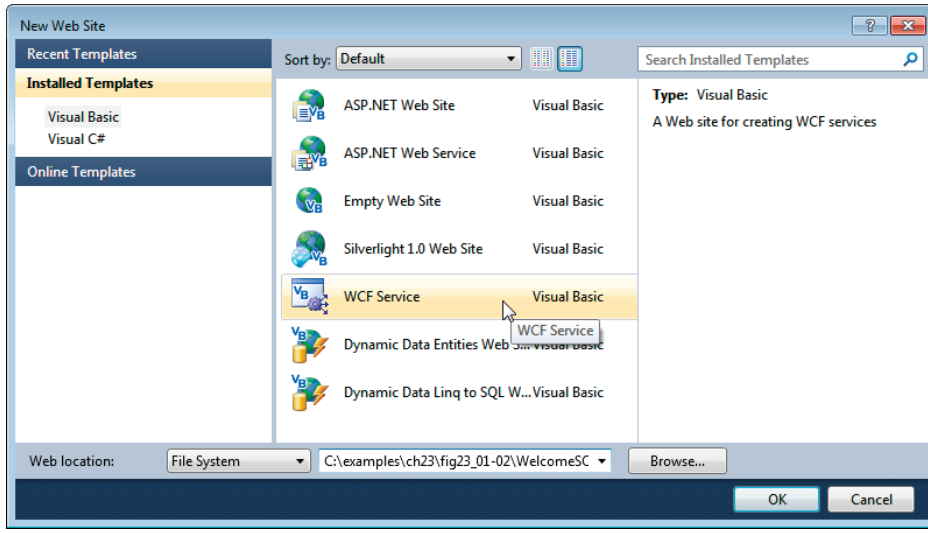


Fig. 23.3 | Creating a **WCF Service** in Visual Web Developer.

the `ServiceContract` and `OperationContract` attributes. In addition, the `IService.vb` file defines a class named `CompositeType` with a `DataContract` attribute (discussed in Section 23.8). The interface contains two sample service methods named `GetData` and `GetDataUsingContract`. The `Service.vb` contains the code that defines these methods.

Step 3: Modifying and Renaming the Code-Behind File

To create the `WelcomeSOAPXMLService` service developed in this section, modify `IService.vb` and `Service.vb` by replacing the sample code provided by Visual Web Developer with the code from the `IWelcomeSOAPXMLService` and `WelcomeSOAPXMLService` files (Figs. 23.1 and 23.2, respectively). Then rename the files to `IWelcomeSOAPXMLService.vb` and `WelcomeSOAPXMLService.vb` by right clicking each file in the Solution Explorer and choosing **Rename**.

Step 4: Examining the SVC File

The `Service.svc` file, when accessed through a web browser, provides information about the web service. However, if you open the SVC file on disk, it contains only

```
<%@ ServiceHost Language="VB" Debug="true" Service="Service"
   CodeBehind="~/App_Code/Service.vb" %>
```

to indicate the programming language in which the web service's code-behind file is written, the `Debug` attribute (enables a page to be compiled for debugging), the name of the service and the code-behind file's location. When you request the SVC page in a web browser, WCF uses this information to dynamically generate the WSDL document.

Step 5: Modifying the SVC File

If you change the code-behind file name or the class name that defines the web service, you must modify the SVC file accordingly. Thus, after defining class `WelcomeSOAP-`

`XMLService` in the code-behind file `WelcomeSOAPXMLService.vb`, modify the SVC file as follows:

```
<%@ ServiceHost Language="VB" Debug="true"
   Service="WelcomeSOAPXMLService"
   CodeBehind="~/App_Code/WelcomeSOAPXMLService.vb" %>
```

23.6.4 Deploying the `WelcomeSOAPXMLService`

You can choose **Build Web Site** from the **Build** menu to ensure that the web service compiles without errors. You can also test the web service directly from Visual Web Developer by selecting **Start Debugging** from the **Debug** menu. The first time you do this, the **Debugging Not Enabled** dialog appears. Click **OK** if you want to enable debugging. Next, a browser window opens and displays information about the service. This information is generated dynamically when the SVC file is requested. Figure 23.4 shows a web browser displaying the `Service.svc` file for the `WelcomeSOAPXMLService` WCF web service. [*Note:* To view the `Service.svc` file, you must set the `.svc` file as the project's start page by right clicking it in **Solution Explorer** and selecting **Set As Start Page**.]

Once the service is running, you can also access the SVC page from your browser by typing a URL of the following form in a web browser:

```
http://localhost:portNumber/virtualPath/Service.svc
```

(See the actual URL in Fig. 23.4.) By default, the ASP.NET Development Server assigns a random port number to each website it hosts. You can change this behavior by going to the **Solution Explorer** and clicking on the project name to view the **Properties** window (Fig. 23.5). Set the **Use dynamic ports** property to **False** and set the **Port number** property to the port number that you want to use, which can be any unused TCP port. Generally, you don't do this for web services that will be deployed to a real web server. You can also change the service's virtual path, perhaps to make the path shorter or more readable.

Web Services Description Language

To consume a web service, a client must determine the service's functionality and how to use it. For this purpose, web services normally contain a **service description**. This is an XML document that conforms to the **Web Service Description Language** (**WSDL**)—an XML vocabulary that defines the methods a web service makes available and how clients interact with them. The WSDL document also specifies lower-level information that clients might need, such as the required formats for requests and responses.

WSDL documents help applications determine how to interact with the web services described in the documents. When viewed in a web browser, an SVC file presents a link to the service's WSDL document and information on using the utility **`svcutil.exe`** to generate test console applications. The `svcutil.exe` tool is included with Visual Studio 2010 and Visual Web Developer. We do not use `svcutil.exe` to test our services, opting instead to build our own test applications. When a client requests the SVC file's URL followed by `?wsdl`, the server autogenerates the WSDL that describes the web service and returns the WSDL document. Copy the SVC URL (which ends with `.svc`) from the browser's address field in Fig. 23.4, as you'll need it in the next section to build the client application. Also, leave the web service running so the client can interact with it.

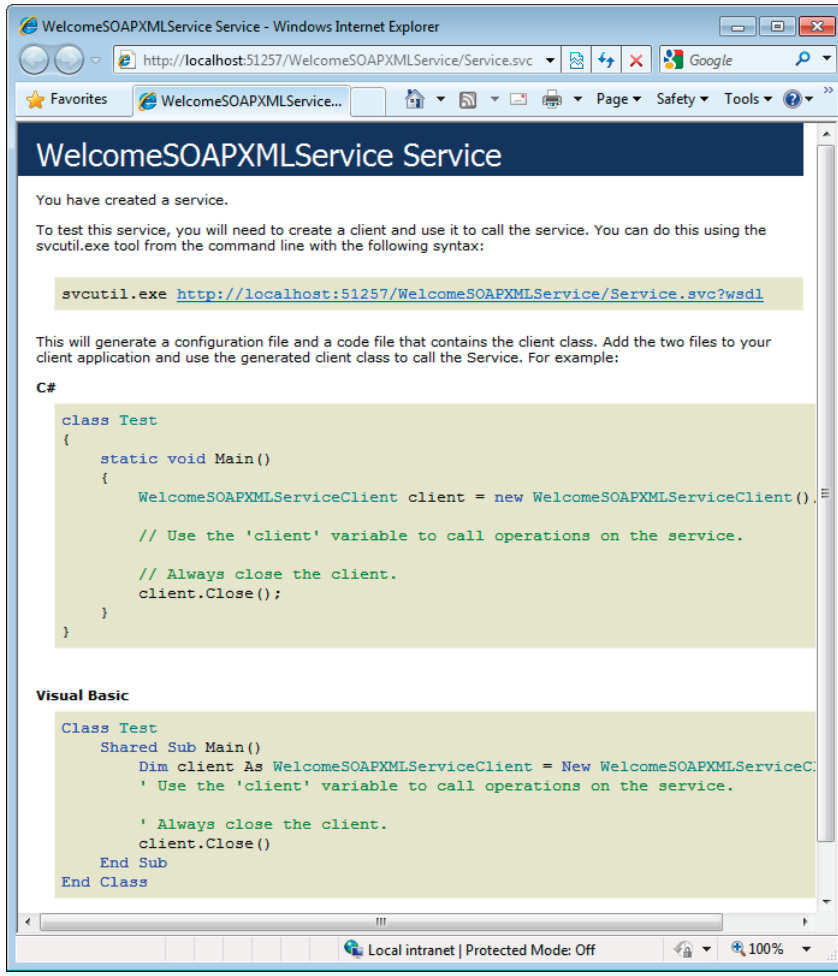


Fig. 23.4 | SVC file rendered in a web browser.

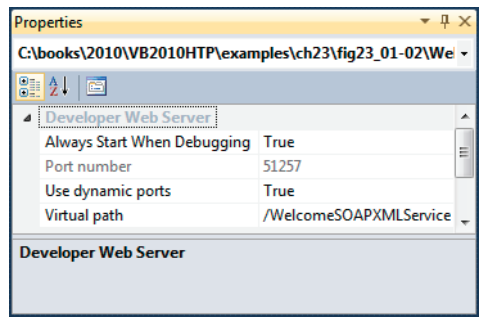


Fig. 23.5 | WCF web service **Properties** window.

23.6.5 Creating a Client to Consume the `WelcomeSOAPXMLService`

Now that you've defined and deployed the web service, let's consume it from a client application. A .NET web-service client can be any type of .NET application, such as a Windows application, a console application or a web application. You can enable a client application to consume a web service by **adding a service reference** to the client. Figure 23.6 diagrams the parts of a client for a SOAP-based web service after a service reference has been added. [*Note*: This section discusses Visual Basic 2010 Express, but the discussion also applies to Visual Web Developer 2010 Express.]

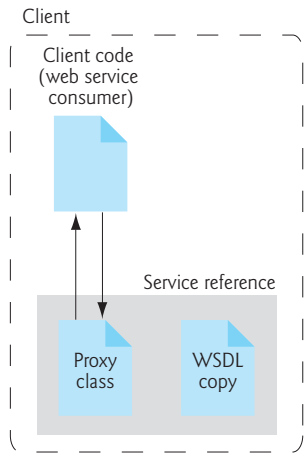


Fig. 23.6 | .NET WCF web service client after a web-service reference has been added.

An application that consumes a SOAP-based web service actually consists of two parts—a proxy class representing the web service and a client application that accesses the web service via a proxy object (that is, an instance of the proxy class). A **proxy class** handles all the "plumbing" required for service method calls (that is, the networking details and the formation of SOAP messages). Whenever the client application calls a web service's method, the application actually calls a corresponding method in the proxy class. This method has the same name and parameters as the web service's method that is being called, but formats the call to be sent as a request in a SOAP message. The web service receives this request as a SOAP message, executes the method call and sends back the result as another SOAP message. When the client application receives the SOAP message containing the response, the proxy class deserializes it and returns the results as the return value of the web-service method that was called. Figure 23.7 depicts the interactions among the client code, proxy class and web service. The proxy class is not shown in the project unless you click the **Show All Files** button in the **Solution Explorer**.

Many aspects of web-service creation and consumption—such as generating WSDL files and proxy classes—are handled by Visual Web Developer, Visual Basic 2010 and WCF. Although developers are relieved of the tedious process of creating these files, they can still modify the files if necessary. This is required only when developing advanced web services—none of our examples require modifications to these files.

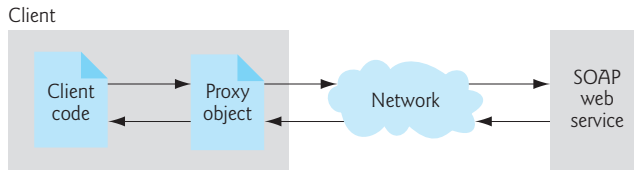


Fig. 23.7 | Interaction between a web-service client and a SOAP web service.

We now create a client and generate a proxy class that allows the client to access the `WelcomeSOAPXMLService` web service. First create a Windows application named `WelcomeSOAPXMLClient` in Visual Basic 2010, then perform the following steps.

Step 1: Opening the **Add Service Reference** *Dialog*
Right click the project name in the **Solution Explorer** and select **Add Service Reference...** to display the **Add Service Reference** dialog.

Step 2: Specifying the Web Service's Location
In the dialog, enter the URL of `WelcomeSOAPXMLService`'s `.svc` file (that is, the URL you copied from Fig. 23.4) in the **Address** field. When you specify the service you want to consume, the IDE accesses the web service's WSDL information and copies it into a WSDL file that is stored in the client project's `Service References` folder. This file is visible when you view all of your project's files in the **Solution Explorer**. [*Note:* A copy of the WSDL file provides the client application with local access to the web service's description. To ensure that the WSDL file is up to date, Visual Basic 2010 provides an **Update Service Reference** option (available by right clicking the service reference in the **Solution Explorer**), which updates the files in the `Service References` folder.]

Many companies that provide web services simply distribute the exact URLs at which their web services can be accessed. The **Add Service Reference** dialog also allows you to search for services on your local machine or on the Internet.

Step 3: Renaming the Service Reference's Namespace
In the **Add Service Reference** dialog, rename the service reference's namespace by changing the **Namespace** field to `ServiceReference`.

Step 4: Adding the Service Reference
Click the **Ok** button to add the service reference.

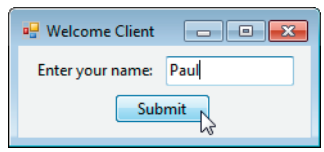
Step 5: Viewing the Service Reference in the **Solution Explorer**
The **Solution Explorer** should now contain a **Service References** folder with a node showing the namespace you specified in *Step 3*.

23.6.6 Consuming the `WelcomeSOAPXMLService`

The application in Fig. 23.8 uses the `WelcomeSOAPXMLService` service to send a welcome message. You are already familiar with Visual Basic applications that use `Labels`, `TextBoxes` and `Buttons`, so we focus our discussions on the web-services concepts in this chapter's applications.

```
1  ' Fig. 23.8: WelcomeSOAPXML.vb
2  ' Client that consumes WelcomeSOAPXMLService.
3  Public Class WelcomeSOAPXML
4     ' reference to web service
5     Private client As New ServiceReference.WelcomeSOAPXMLServiceClient()
6
7     ' creates welcome message from text input and web service
8     Private Sub submitButton_Click(ByVal sender As System.Object,
9        ByVal e As System.EventArgs) Handles submitButton.Click
10
11       MessageBox.Show(client.Welcome(textBox.Text))
12    End Sub ' submitButton_Click
13 End Class ' WelcomeSOAPXML
```

a) User inputs name



b) Message sent from **WelcomeSOAPXML-Service**

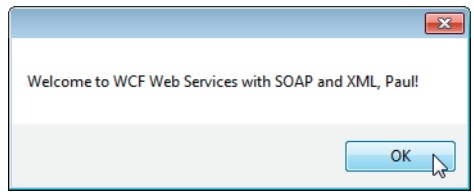


Fig. 23.8 | Client that consumes `WelcomeSOAPXMLService`.

Line 5 defines a new `ServiceReference.WelcomeSOAPXMLServiceClient` proxy object named `client`. The event handler uses this object to call methods of the `WelcomeSOAPXMLService` web service. Line 11 invokes the `WelcomeSOAPXMLService` web service's `Welcome` method. The call is made via the local proxy object `client`, which then communicates with the web service on the client's behalf. If you downloaded the example from `www.deitel.com/books/vb2010htp/`, you may need to regenerate the proxy by removing the service reference, then adding it again, because ASP.NET Development Server may use a different port number on your computer. To do so, right click `ServiceReference` in the **Service References** folder in the **Solution Explorer** and select option **Delete**. Then follow the instructions in Section 23.6.5 to add the service reference to the project.

When the application runs, enter your name and click the **Submit** button. The application invokes the `Welcome` service method to perform the appropriate task and return the result, then displays the result in a `MessageBox`.

23.7 Publishing and Consuming REST-Based XML Web Services

In the previous section, we used a proxy object to pass data to and from a WCF web service using the SOAP protocol. In this section, we access a WCF web service using the REST

architecture. We modify the `IWelcomeSOAPXMLService` example to return data in plain XML format. You can create a **WCF Service** project as you did in Section 23.6 to begin.

23.7.1 HTTP `get` and `post` Requests

The two most common **HTTP request types** (also known as **request methods**) are `get` and `post`. A **`get` request** typically gets (or retrieves) information from a server. Common uses of `get` requests are to retrieve a document or an image, or to fetch search results based on a user-submitted search term. A **`post` request** typically posts (or sends) data to a server. Common uses of `post` requests are to send form data or documents to a server.

An HTTP request often posts data to a **server-side form handler** that processes the data. For example, when a user performs a search or participates in a web-based survey, the web server receives the information specified in the XHTML form as part of the request. *Both* types of requests can be used to send form data to a web server, yet each request type sends the information differently.

A `get` request sends information to the server in the URL. For example, in the following URL

```
www.google.com/search?q=deitel
```

`search` is the name of Google's server-side form handler, `q` is the `name` of a *variable* in Google's search form and `deitel` is the search term. A `?` separates the **query string** from the rest of the URL in a request. A *name*/*value* pair is passed to the server with the *name* and the *value* separated by an equals sign (`=`). If more than one *name*/*value* pair is submitted, each pair is separated by an ampersand (`&`). The server uses data passed in a query string to retrieve an appropriate resource from the server. The server then sends a **response** to the client. A `get` request may be initiated by submitting an XHTML form whose `method` attribute is set to `"get"`, or by typing the URL (possibly containing a query string) directly into the browser's address bar.

A `post` request sends form data as part of the HTTP message, not as part of the URL. A `get` request typically limits the query string (that is, everything to the right of the `?`) to a specific number of characters. For example, Internet Explorer restricts the entire URL to no more than 2083 characters. Typically, large amounts of information should be sent using the `post` method. The `post` method is also sometimes preferred because it *hides* the submitted data from the user by embedding it in an HTTP message. If a form submits hidden input values along with user-submitted data, the `post` method might generate a URL like `www.searchengine.com/search`. The form data still reaches the server for processing, but the user does not see the exact information sent.

23.7.2 Creating a REST-Based XML WCF Web Service

Step 1: Adding the **WebGet** *Attribute*

`IWelcomeRESTXMLService` interface (Fig. 23.9) is a modified version of the `IWelcomeSOAPXMLService` interface. The `Welcome` method's **`WebGet`** attribute (line 10) maps a method to a unique URL that can be accessed via an HTTP `get` operation programmatically or in a web browser. To use the `WebGet` attribute, we import the `System.ServiceModel.Web` namespace (line 4). `WebGet`'s **`UriTemplate`** property (line 10) specifies the URI format that is used to invoke the method. You can access the `Welcome` method in a web

browser by appending text that matches the `UriTemplate` definition to the end of the service's location, as in `http://localhost:51424/WelcomeRESTXMLService/Service.svc/welcome/Bruce`. `WelcomeRESTXMLService` (Fig. 23.10) is the class that implements the `IWelcomeRESTXMLService` interface; it is similar to the `WelcomeSOAPXMLService` class (Fig. 23.2).

```
1  ' Fig. 23.9: IWelcomeRESTXMLService.vb
2  ' WCF web-service interface. A class that implements this interface
3  ' returns a welcome message through REST architecture and XML data format.
4  Imports System.ServiceModel.Web
5
6  <ServiceContract()>
7  Public Interface IWelcomeRESTXMLService
8     ' returns a welcome message
9     <OperationContract()>
10    <WebGet(UriTemplate:="welcome/{yourName}")>
11    Function Welcome(ByVal yourName As String) As String
12 End Interface ' IWelcomeRESTXMLService
```

Fig. 23.9 | WCF web-service interface. A class that implements this interface returns a welcome message through REST architecture and XML data format.

```
1  ' Fig. 23.10: WelcomeRESTXMLService.vb
2  ' WCF web service that returns a welcome message using REST architecture
3  ' and XML data format.
4  Public Class WelcomeRESTXMLService
5     Implements IWelcomeRESTXMLService
6
7     ' returns a welcome message
8     Public Function Welcome(ByVal yourName As String) _
9        As String Implements IWelcomeRESTXMLService.Welcome
10
11       Return "Welcome to WCF Web Services with REST and XML, " &
12          yourName & "!"
13    End Function ' Welcome
14 End Class ' WelcomeRESTXMLService
```

Fig. 23.10 | WCF web service that returns a welcome message using REST architecture and XML data format.

Step 2: Modifying the `Web.config` *File*

Figure 23.11 shows part of the default `Web.config` file modified to use REST architecture. The `endpointBehaviors` element (lines 16–20) in the `behaviors` element indicates that this web service endpoint will be accessed using the web programming model (REST). The nested `webHttp` element specifies that clients communicate with this service using the standard HTTP request/response mechanism. The `protocolMapping` element (lines 22–24) in the `system.serviceModel` element, changes the default protocol for communicating with this web service (normally SOAP) to the `webHttpBinding`, which is used for REST-based HTTP requests.

```
<system.serviceModel>
   <behaviors>
      <serviceBehaviors>
         <behavior>
            <!-- To avoid disclosing metadata information, set the
               value below to false and remove the metadata
               endpoint above before deployment -->
            <serviceMetadata httpGetEnabled="true"/>
            <!-- To receive exception details in faults for debugging
               purposes, set the value below to true.  Set to false
               before deployment to avoid disclosing exception
               information -->
            <serviceDebug includeExceptionDetailInFaults="false"/>
         </behavior>
      </serviceBehaviors>
      <endpointBehaviors>
         <behavior>
            <webHttp/>
         </behavior>
      </endpointBehaviors>
   </behaviors>
   <protocolMapping>
      <add scheme="http" binding="webHttpBinding"/>
   </protocolMapping>
</system.serviceModel>
```

Fig. 23.11 | `WelcomeRESTXMLService` `Web.config` file.

Figure 23.12 tests the `WelcomeRESTXMLService`'s `Welcome` method in a web browser. The URL specifies the location of the `Service.svc` file and uses the URI template to invoke method `Welcome` with the argument `Bruce`. The browser displays the XML data response from `WelcomeRESTXMLService`. Next, you'll learn how to consume this service.

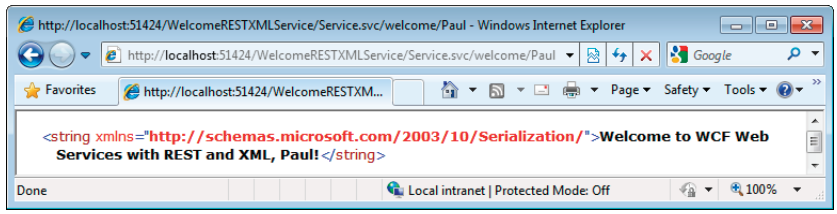


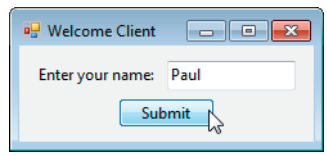
Fig. 23.12 | Response from `WelcomeRESTXMLService` in XML data format.

23.7.3 Consuming a REST-Based XML WCF Web Service

`WelcomeRESTXML` (Fig. 23.13) uses the **`WebClient`** class to invoke the web service and receive its response. In line 5, we import the XML message's namespace (seen in Fig. 23.12), which is required to parse the service's XML response. The keyword `With-Events` in line 9 indicates that the `WebClient` object has events associated with it and enables you to use the variable's name in an event handler's `Handles` clause.

```
1  ' Fig. 23.13: WelcomeRESTXML.vb
2  ' Client that consumes the WelcomeRESTXMLService.
3  Imports System.Net
4  Imports System.Xml.Linq
5  Imports <xmlns="http://schemas.microsoft.com/2003/10/Serialization/">
6
7  Public Class WelcomeRESTXML
8     ' object to invoke the WelcomeRESTXMLService
9     Private WithEvents service As New WebClient()
10
11    ' get user input and pass it to the web service
12    Private Sub submitButton_Click(ByVal sender As System.Object,
13       ByVal e As System.EventArgs) Handles submitButton.Click
14
15       ' send request to WelcomeRESTXMLService
16       service.DownloadStringAsync(New Uri(
17          "http://localhost:51424/WelcomeRESTXMLService/Service.svc/" &
18          "welcome/" & textBox.Text))
19    End Sub ' submitButton_Click
20
21    ' process web-service response
22    Private Sub service_DownloadStringCompleted(ByVal sender As Object,
23       ByVal e As System.Net.DownloadStringCompletedEventArgs) _
24       Handles service.DownloadStringCompleted
25
26       ' check if any errors occurred in retrieving service data
27       If e.Error Is Nothing Then
28          ' parse the returned XML string (e.Result)
29          Dim xmlResponse = XDocument.Parse(e.Result)
30
31          ' use XML axis property to access the <string> element's value
32          MessageBox.Show(xmlResponse.<string>.Value)
33       End If
34    End Sub ' service_DownloadStringCompleted
35 End Class ' WelcomeRESTXML
```

a) User inputs name.



b) Message sent from **WelcomeRESTXMLService**.

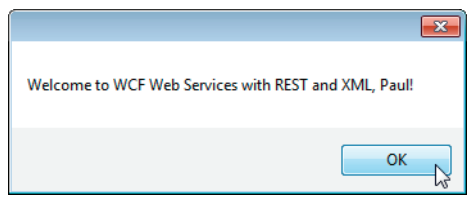


Fig. 23.13 | Client that consumes the `WelcomeRESTXMLService`.

In this example, we process the `WebClient`'s **`DownloadStringCompleted`** event, which occurs when the client receives the completed response from the web service. Line 16 calls the `service` object's **`DownloadStringAsync`** method to invoke the web service asynchronously. (There is also a synchronous `DownloadString` method that does not return until it receives the response.) The method's argument (that is, the URL to invoke the web service) must be specified as an object of class **`Uri`**. Class `Uri`'s constructor receives a `String` representing a uniform resource identifier. [*Note:* The URL's port number must match the one issued to the web service by the ASP.NET Development Server.] When the call to the web service completes, the `WebClient` object raises the `DownloadStringCompleted` event. Its event handler has a parameter `e` of type **`DownloadStringCompletedEventArgs`** which contains the information returned by the web service. We can use this variable's properties to get the returned XML document (**`e.Result`**) and any errors that may have occurred during the process (**`e.Error`**). We then parse the XML response using `XDocument` method `Parse` (line 29) and display our welcome `String` in a `MessageBox` (line 32).

23.8 Publishing and Consuming REST-Based JSON Web Services

We now build a RESTful web service that returns data in JSON format.

23.8.1 Creating a REST-Based JSON WCF Web Service

By default, a web-service method with the `WebGet` attribute returns data in XML format. In Fig. 23.14, we modify the `WelcomeRESTXMLService` to return data in JSON format by setting `WebGet`'s **`ResponseFormat`** property to `WebMessageFormat.Json` (line 10). (`WebMessageFormat.XML` is the default value.) For JSON serialization to work properly, the objects being converted to JSON must have `Public` properties. This enables the JSON serialization to create name/value pairs representing each `Public` property and its corresponding value. The previous examples return `String` objects containing the responses. Even though `String`s are objects, `String`s do not have any `Public` properties that represent their contents. So, lines 17–30 define a `TextMessage` class that encapsulates a `String` value and defines a `Public` property `Message` to access that value. The **`DataContract`** attribute (line 16) exposes the `TextMessage` class to the client access. Similarly, the **`DataMember`** attribute exposes a property of this class to the client. This property will appear in the JSON object as a name/value pair. Only `DataMembers` of a `DataContract` are serialized.

```
1  ' Fig. 23.14: IWelcomeRESTJSONService.vb
2  ' WCF web-service interface that returns a welcome message through REST
3  ' architecture and JSON format.
4  Imports System.ServiceModel.Web
5
6  <ServiceContract()>
7  Public Interface IWelcomeRESTJSONService
```

Fig. 23.14 | WCF web-service interface that returns a welcome message through REST architecture and JSON format. (Part 1 of 2.)

```
8      ' returns a welcome message
9      <OperationContract()>
10     <WebGet(ResponseFormat:=WebMessageFormat.Json,
11        UriTemplate:="welcome/{yourName}")>
12     Function Welcome(ByVal yourName As String) As TextMessage
13  End Interface ' IWelcomeRESTJSONService
14
15  ' class to encapsulate a String to send in JSON format
16  <DataContract()>
17  Public Class TextMessage
18     Public messageValue As String
19
20     ' property Message
21     <DataMember()>
22     Public Property Message() As String
23        Get
24           Return messageValue
25        End Get
26        Set(ByVal value As String)
27           messageValue = value
28        End Set
29     End Property ' Message
30  End Class ' TextMessage
```

Fig. 23.14 | WCF web-service interface that returns a welcome message through REST architecture and JSON format. (Part 2 of 2.)

Figure 23.15 shows the implementation of the interface of Fig. 23.14. The `Welcome` method (lines 8–15) returns a `TextMessage` object, reflecting the changes we made to the interface class. This object is automatically serialized in JSON format (as a result of line 10 in Fig. 23.14) and sent to the client.

```
1   ' Fig. 23.15: WelcomeRESTJSONService.vb
2   ' WCF web service that returns a welcome message through REST architecture
3   ' and JSON format.
4   Public Class WelcomeRESTJSONService
5      Implements IWelcomeRESTJSONService
6
7      ' returns a welcome message
8      Public Function Welcome(ByVal yourName As String)
9         As TextMessage Implements IWelcomeRESTJSONService.Welcome
10        ' add welcome message to field of TextMessage object
11        Dim welcomeString As New TextMessage
12        welcomeString.Message = "Welcome to WCF Web Services with REST " &
13           "and JSON, " & yourName & "!"
14        Return welcomeString
15     End Function ' Welcome
16  End Class ' WelcomeRESTJSONService
```

Fig. 23.15 | WCF web service that returns a welcome message through REST architecture and JSON format.

We can once again test the web service using a web browser, by accessing the `Service.svc` file (`http://localhost:49745/WelcomeRESTJSONService/Service.svc`) and appending the URI template (`welcome/`*yourName*) to the address. The response prompts you to download a file called *yourName*, which is a text file. If you save it to disk, the file will have the `.json` extension. This contains the JSON formatted data. By opening the file in a text editor such as Notepad (Fig. 23.16), you can see the service response as a JSON object. Notice that the property named `Message` has the welcome message as its value.

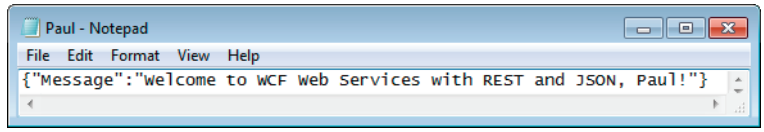


Fig. 23.16 | Response from `WelcomeRESTJSONService` in JSON data format.

23.8.2 Consuming a REST-Based JSON WCF Web Service

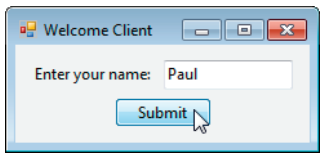
We mentioned earlier that all types passed to and from web services can be supported by REST. Custom types that are sent to or from a REST web service are converted to XML or JSON data format. This process is referred to as **XML serialization** or **JSON serialization**, respectively. In Fig. 23.17, we consume the `WelcomeRESTJSONService` service using an object of the `System.Runtime.Serialization.Json` library's **`DataContractJsonSerializer`** class (lines 30–31). To use the `System.Runtime.Serialization.Json` library and `DataContractJsonSerializer` class, you must include a reference to the `System.ServiceModel.Web` and `System.Runtime.Serialization` assemblies in the project. To do so, right click the project name, select **Add Reference** and add the `System.ServiceModel.Web` and `System.Runtime.Serialization` assemblies. The `TextMessage` class (lines 43–45) maps the JSON response's fields for the `DataContractJsonSerializer` to deserialize. We add the **`Serializable`** attribute (line 42) to the `TextMessage` class to recognize it as a valid serializable object we can convert to and from JSON format. Also, this class on the client must have `Public` data or properties that match the `Public` data or properties in the corresponding class from the web service. Since we want to convert the JSON response into a `TextMessage` object, we set the `DataContractJsonSerializer`'s type parameter to `TextMessage` (line 31). In line 33, we use the `System.Text` namespace's `Encoding.Unicode.GetBytes` method to convert the JSON response to a Unicode encoded byte array, and encapsulate the byte array in a `MemoryStream` object so we can read data from the array using stream semantics. The bytes in the `MemoryStream` object are read by the `DataContractJsonSerializer` and deserialized into a `TextMessage` object (line 32).

```
1  ' Fig. 23.17: WelcomeRESTJSON.vb
2  ' Client that consumes WelcomeRESTJSONService.
3  Imports System.IO
4  Imports System.Net
5  Imports System.Runtime.Serialization.Json
6  Imports System.Text
```

Fig. 23.17 | Client that consumes `WelcomeRESTJSONService`. (Part 1 of 2.)

```
7
8  Public Class WelcomeRESTJSON
9     ' object to invoke the WelcomeRESTJSONService
10    Private WithEvents service As New WebClient()
11
12    ' creates welcome message from text input and web service
13    Private Sub submitButton_Click(ByVal sender As System.Object,
14       ByVal e As System.EventArgs) Handles submitButton.Click
15
16       ' send request to WelcomeRESTJSONService
17       service.DownloadStringAsync(New Uri(
18          "http://localhost:49745/WelcomeRESTJSONService/Service.svc/" &
19          "welcome/" & textBox.Text))
20    End Sub ' submitButton
21
22    ' process web-service response
23    Private Sub service_DownloadStringCompleted(ByVal sender As Object,
24       ByVal e As System.Net.DownloadStringCompletedEventArgs) _
25       Handles service.DownloadStringCompleted
26
27       ' check if any errors occurred in retrieving service data
28       If e.Error Is Nothing Then
29          ' deserialize response into a TextMessage object
30          Dim JSONSerializer _
31             As New DataContractJsonSerializer(GetType(TextMessage))
32          Dim welcomeString = JSONSerializer.ReadObject(
33             New MemoryStream(Encoding.Unicode.GetBytes(e.Result)))
34
35          ' display Message text
36          MessageBox.Show(CType(welcomeString, TextMessage).Message)
37       End If
38    End Sub ' service_DownloadStringCompleted
39 End Class ' WelcomeRESTJSON
40
41 ' TextMessage class representing a JSON object
42 <Serializable()>
43 Public Class TextMessage
44    Public Message As String
45 End Class ' TextMessage
```

a) User inputs name.



b) Message sent from `WelcomeRESTJSONService`.

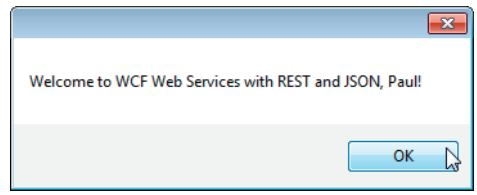


Fig. 23.17 | Client that consumes `WelcomeRESTJSONService`. (Part 2 of 2.)

23.9 Blackjack Web Service: Using Session Tracking in a SOAP-Based WCF Web Service

In Chapter 13, we described the advantages of maintaining information about users to personalize their experiences. In particular, we discussed session tracking using cookies and `HttpSessionState` objects. Next, we incorporate session tracking into a SOAP-based WCF web service.

Suppose a client application needs to call several methods from the same web service, possibly several times each. In such a case, it can be beneficial for the web service to maintain state information for the client. Session tracking eliminates the need for information about the client to be passed between the client and the web service multiple times. For example, a web service providing access to local restaurant reviews would benefit from storing the client user's street address. Once the user's address is stored in a session variable, web service methods can return personalized, localized results without requiring that the address be passed in each method call. This not only improves performance but also requires less effort on your part—less information is passed in each method call.

23.9.1 Creating a Blackjack Web Service

Web services store session information to provide more intuitive functionality. Our next example is a SOAP-based web service that assists programmers in developing a blackjack card game. The web service provides methods to deal a card and to evaluate a hand of cards. After presenting the web service, we use it to serve as the dealer for a game of blackjack. The blackjack web service creates a session variable to maintain a unique deck of cards for each client application. Several clients can use the service at the same time, but method calls made by a specific client use only the deck stored in that client's session. Our example uses a simple subset of casino blackjack rules:

> *Two cards each are dealt to the dealer and the player. The player's cards are dealt face up. Only the dealer's first card is dealt face up. Each card has a value. A card numbered 2 through 10 is worth its face value. Jacks, queens and kings each count as 10. Aces can count as 1 or 11—whichever value is more beneficial to the player (as we'll soon see). If the sum of the player's two initial cards is 21 (that is, the player was dealt a card valued at 10 and an ace, which counts as 11 in this situation), the player has "blackjack" and immediately wins the game. Otherwise, the player can begin taking additional cards one at a time. These cards are dealt face up, and the player decides when to stop taking cards. If the player "busts" (that is, the sum of the player's cards exceeds 21), the game is over, and the player loses. When the player is satisfied with the current set of cards, the player "stays" (that is, stops taking cards), and the dealer's hidden card is revealed. If the dealer's total is 16 or less, the dealer must take another card; otherwise, the dealer must stay. The dealer must continue to take cards until the sum of the dealer's cards is greater than or equal to 17. If the dealer exceeds 21, the player wins. Otherwise, the hand with the higher point total wins. If the dealer and the player have the same point total, the game is a "push" (that is, a tie), and no one wins.*

The Blackjack WCF web service's interface (Fig. 23.18) uses a `ServiceContract` with the `SessionMode` property set to `Required` (line 3). This means the service requires sessions to execute correctly. By default, the `SessionMode` property is set to `Allowed`. It can also be set to `NotAllowed` to disable sessions.

```
1  ' Fig. 23.18: IBlackjackService.vb
2  ' Blackjack game WCF web-service interface.
3  <ServiceContract(SessionMode:=SessionMode.Required)> _
4  Public Interface IBlackjackService
5     ' deals a card that has not been dealt
6     <OperationContract()>
7     Function DealCard() As String
8
9     ' creates and shuffles the deck
10    <OperationContract()>
11    Sub Shuffle()
12
13    ' calculates value of a hand
14    <OperationContract()>
15    Function GetHandValue(ByVal dealt As String) As Integer
16 End Interface ' IBlackjackService
```

Fig. 23.18 | Blackjack game WCF web-service interface.

The web-service class (Fig. 23.19) provides methods to deal a card, shuffle the deck and determine the point value of a hand. For this example, we want a separate object of the `BlackjackService` class to handle each client session, so we can maintain a unique deck for each client. To do this, we must specify this behavior in the **`ServiceBehavior`** attribute (line 5). Setting the `ServiceBehavior`'s **`InstanceContextMode`** property to `PerSession` creates a new instance of the class for each session. The `InstanceContextMode` property can also be set to `PerCall` or `Single`. `PerCall` uses a new object of the web-service class to handle every method call to the service. `Single` uses the same object of the web-service class to handle all calls to the service.

```
1  ' Fig. 23.19: BlackjackService.vb
2  ' Blackjack game WCF web service.
3  Imports System.Collections.Generic
4
5  <ServiceBehavior(InstanceContextMode:=InstanceContextMode.PerSession)>
6  Public Class BlackjackService
7     Implements IBlackjackService
8     ' create persistent session deck-of-cards object
9     Dim deck As New List(Of String)
10
11    ' deals card that has not yet been dealt
12    Public Function DealCard() As String _
13       Implements IBlackjackService.DealCard
14
15       Dim card As String = Convert.ToString(deck(0)) ' get first card
16       deck.RemoveAt(0) ' remove card from deck
17       Return card
18    End Function ' DealCard
19
```

Fig. 23.19 | Blackjack game WCF web service. (Part 1 of 3.)

```
20      ' creates and shuffles a deck of cards
21      Public Sub Shuffle() Implements IBlackjackService.Shuffle
22         Dim randomObject As New Random() ' generates random numbers
23
24         deck.Clear() ' clears deck for new game
25
26         ' generate all possible cards
27         For face = 1 To 13 ' loop through face values
28            For suit As Integer = 0 To 3 ' loop through suits
29               deck.Add(face & " " & suit) ' add card (string) to deck
30            Next suit
31         Next face
32
33         ' shuffles deck by swapping each card with another card randomly
34         For i = 0 To deck.Count - 1
35            ' get random index
36            Dim newIndex = randomObject.Next(deck.Count - 1)
37            Dim temporary = deck(i) ' save current card in temporary variable
38            deck(i) = deck(newIndex) ' copy randomly selected card
39            deck(newIndex) = temporary ' copy current card back into deck
40         Next
41      End Sub ' Shuffle
42
43      ' computes value of hand
44      Public Function GetHandValue(ByVal dealt As String) As Integer _
45         Implements IBlackjackService.GetHandValue
46         ' split string containing all cards
47         Dim tab As Char = Convert.ToChar(vbTab)
48         Dim cards As String() = dealt.Split(tab) ' get array of cards
49         Dim total As Integer = 0 ' total value of cards in hand
50         Dim face As Integer ' face of the current card
51         Dim aceCount As Integer = 0 ' number of aces in hand
52
53         ' loop through the cards in the hand
54         For Each card In cards
55            ' get face of card
56            face = Convert.ToInt32(card.Substring(0, card.IndexOf(" ")))
57
58            Select Case face
59               Case 1 ' if ace, increment aceCount
60                  aceCount += 1
61               Case 11 To 13 ' if jack, queen or king add 10
62                  total += 10
63               Case Else ' otherwise, add value of face
64                  total += face
65            End Select
66         Next
67
68         ' if there are any aces, calculate optimum total
69         If aceCount > 0 Then
70            ' if it is possible to count one ace as 11, and the rest
71            ' as 1 each, do so; otherwise, count all aces as 1 each
```

Fig. 23.19 | Blackjack game WCF web service. (Part 2 of 3.)

```
72            If (total + 11 + aceCount - 1 <= 21) Then
73               total += 11 + aceCount - 1
74            Else
75               total += aceCount
76            End If
77         End If
78
79         Return total
80      End Function ' GetHandValue
81   End Class ' BlackjackService
```

Fig. 23.19 | Blackjack game WCF web service. (Part 3 of 3.)

We represent each card as a `String` consisting of a digit (that is, `1`–`13`) representing the card's face (for example, ace through king), followed by a space and a digit (that is, `0`–`3`) representing the card's suit (for example, clubs, diamonds, hearts or spades). For example, the jack of hearts is represented as `"11 2"`, and the two of clubs as `"2 0"`. After deploying the web service, we create a Windows `Forms` application that uses the `BlackjackService`'s methods to implement a blackjack game.

Method `DealCard` (lines 12–18) removes a card from the deck and sends it to the client. Without using session tracking, the deck of cards would need to be passed back and forth with each method call. Using session state makes the method easy to call (it requires no arguments) and avoids the overhead of sending the deck over the network multiple times.

Method `DealCard` (lines 12–18) manipulates the current user's deck (the `List` of `String`s defined at line 9). From the user's deck, `DealCard` obtains the current top card (line 15), removes the top card from the deck (line 16) and returns the card's value as a `String` (line 17).

Method `Shuffle` (lines 21–41) fills the `List` object representing a deck of cards and shuffles it. Lines 27–31 generate `String`s in the form "*face suit*" to represent each card in a deck. Lines 34–40 shuffle the deck by swapping each card with another randomly selected card in the deck.

Method `GetHandValue` (lines 44–80) determines the total value of cards in a hand by trying to attain the highest score possible without going over 21. Recall that an ace can be counted as either 1 or 11, and all face cards count as 10.

As you'll see in Fig. 23.20, the client application maintains a hand of cards as a `String` in which each card is separated by a tab character. Line 48 of Fig. 23.19 tokenizes the hand of cards (represented by `dealt`) into individual cards by calling `String` method `Split` and passing to it the tab character. `Split` uses the delimiter characters to separate tokens in the `String`. Lines 54–66 count the value of each card. Line 56 retrieves the first integer—the face—and uses that value in the `Select Case` statement (lines 58–65). If the card is an ace, the method increments variable `aceCount` (line 60). We discuss how this variable is used shortly. If the card is an 11, 12 or 13 (jack, queen or king), the method adds 10 to the total value of the hand (line 62). If the card is anything else, the method increases the total by that value (line 64).

Because an ace can represent 1 or 11, additional logic is required to process aces. Lines 69–77 process the aces after all the other cards. If a hand contains several aces, only one ace can be counted as 11 (if two aces each are counted as 11, the hand would have a losing value of at least 22). The condition in line 72 determines whether counting one ace as 11

and the rest as 1 results in a total that does not exceed 21. If this is possible, line 73 adjusts the total accordingly. Otherwise, line 75 adjusts the total, counting each ace as 1.

Method `GetHandValue` maximizes the value of the current cards without exceeding 21. Imagine, for example, that the dealer has a 7 and receives an ace. The new total could be either 8 or 18. However, `GetHandValue` always maximizes the value of the cards without going over 21, so the new total is 18.

Modifying the `web.config` *File*

To allow this web service to perform session tracking, you must modify the web.config file to include the following element in the `system.serviceModel` element:s

```
<protocolMapping>
<add scheme="http" binding="wsHttpBinding"/>
</protocolMapping>
```

23.9.2 Consuming the Blackjack Web Service

Now we use our blackjack web service in a Windows application (Fig. 23.20). This application uses an instance of `BlackjackServiceClient` (declared in line 7 and created in line 30) to represent the dealer. The web service keeps track of the player's and the dealer's cards (that is, all the cards that have been dealt). As in Section 23.6.5, you must add a service reference to your project so it can access the web service. The code and images for this example are provided with the chapter's examples, which can be downloaded from our website `www.deitel.com/books/vb2010htp`.

```
1  ' Fig. 23.20: Blackjack.vb
2  ' Blackjack game that uses the BlackjackService web service.
3  Imports System.Net
4
5  Public Class Blackjack
6     ' reference to web service
7     Private dealer As ServiceReference.BlackJackServiceClient
8
9     ' string representing the dealer's cards
10    Private dealersCards As String
11
12    ' string representing the player's cards
13    Private playersCards As String
14    Private cardBoxes As List(Of PictureBox) ' list of card images
15    Private currentPlayerCard As Integer ' player's current card number
16    Private currentDealerCard As Integer ' dealer's current card number
17
18    ' enum representing the possible game outcomes
19    Public Enum GameStatus
20       PUSH ' game ends in a tie
21       LOSE ' player loses
22       WIN ' player wins
23       BLACKJACK ' player has blackjack
24    End Enum ' GameStatus
25
```

Fig. 23.20 | Blackjack game that uses the `BlackjackService` web service. (Part 1 of 8.)

```
26      ' sets up the game
27      Private Sub Blackjack_Load(ByVal sender As Object,
28         ByVal e As System.EventArgs) Handles Me.Load
29         ' instantiate object allowing communication with web service
30         dealer = New ServiceReference.BlackJackServiceClient()
31
32         cardBoxes = New List(Of PictureBox)
33
34         ' put PictureBoxes into cardBoxes List
35         cardBoxes.Add(pictureBox1)
36         cardBoxes.Add(pictureBox2)
37         cardBoxes.Add(pictureBox3)
38         cardBoxes.Add(pictureBox4)
39         cardBoxes.Add(pictureBox5)
40         cardBoxes.Add(pictureBox6)
41         cardBoxes.Add(pictureBox7)
42         cardBoxes.Add(pictureBox8)
43         cardBoxes.Add(pictureBox9)
44         cardBoxes.Add(pictureBox10)
45         cardBoxes.Add(pictureBox11)
46         cardBoxes.Add(pictureBox12)
47         cardBoxes.Add(pictureBox13)
48         cardBoxes.Add(pictureBox14)
49         cardBoxes.Add(pictureBox15)
50         cardBoxes.Add(pictureBox16)
51         cardBoxes.Add(pictureBox17)
52         cardBoxes.Add(pictureBox18)
53         cardBoxes.Add(pictureBox19)
54         cardBoxes.Add(pictureBox20)
55         cardBoxes.Add(pictureBox21)
56         cardBoxes.Add(pictureBox22)
57      End Sub ' Blackjack_Load
58
59      ' deals cards to dealer while dealer's total is less than 17,
60      ' then computes value of each hand and determines winner
61      Private Sub DealerPlay()
62         ' reveal dealer's second card
63         Dim tab As Char = Convert.ToChar(vbTab)
64         Dim cards As String() = dealersCards.Split(tab)
65         DisplayCard(1, cards(1))
66
67         Dim nextCard As String
68
69         ' while value of dealer's hand is below 17,
70         ' dealer must take cards
71         While dealer.GetHandValue(dealersCards) < 17
72            nextCard = dealer.DealCard() ' deal new card
73            dealersCards &= vbTab & nextCard
74
75            ' update GUI to show new card
76            MessageBox.Show("Dealer takes a card")
77            DisplayCard(currentDealerCard, nextCard)
```

Fig. 23.20 | Blackjack game that uses the `BlackjackService` web service. (Part 2 of 8.)

```
78             currentDealerCard += 1
79          End While
80
81          Dim dealerTotal As Integer = dealer.GetHandValue(dealersCards)
82          Dim playerTotal As Integer = dealer.GetHandValue(playersCards)
83
84          ' if dealer busted, player wins
85          If dealerTotal > 21 Then
86             GameOver(GameStatus.WIN)
87          Else
88             ' if dealer and player have not exceeded 21,
89             ' higher score wins; equal scores is a push.
90             If dealerTotal > playerTotal Then ' player loses game
91                GameOver(GameStatus.LOSE)
92             ElseIf playerTotal > dealerTotal Then ' player wins game
93                GameOver(GameStatus.WIN)
94             Else ' player and dealer tie
95                GameOver(GameStatus.PUSH)
96             End If
97          End If
98       End Sub ' DealerPlay
99
100      ' displays card represented by cardValue in specified PictureBox
101      Public Sub DisplayCard(
102         ByVal card As Integer, ByVal cardValue As String)
103         ' retrieve appropriate PictureBox
104         Dim displayBox As PictureBox = cardBoxes(card)
105
106         ' if string representing card is empty,
107         ' set displayBox to display back of card
108         If String.IsNullOrEmpty(cardValue) Then
109            displayBox.Image =
110               Image.FromFile("blackjack_images/cardback.png")
111            Return
112         End If
113
114         ' retrieve face value of card from cardValue
115         Dim face As String =
116            cardValue.Substring(0, cardValue.IndexOf(" "))
117
118         ' retrieve the suit of the card from cardValue
119         Dim suit As String =
120            cardValue.Substring(cardValue.IndexOf(" ") + 1)
121
122         Dim suitLetter As Char ' suit letter used to form image file name
123
124         ' determine the suit letter of the card
125         Select Case Convert.ToInt32(suit)
126            Case 0 ' clubs
127               suitLetter = "c"c
128            Case 1 ' diamonds
129               suitLetter = "d"c
```

Fig. 23.20 | Blackjack game that uses the `BlackjackService` web service. (Part 3 of 8.)

```
130          Case 2 ' hearts
131             suitLetter = "h"c
132          Case Else ' spades
133             suitLetter = "s"c
134       End Select
135
136       ' set displayBox to display appropriate image
137       displayBox.Image = Image.FromFile(
138          "blackjack_images/" & face & suitLetter & ".png")
139    End Sub ' DisplayCard
140
141    ' displays all player cards and shows
142    ' appropriate game status message
143    Public Sub GameOver(ByVal winner As GameStatus)
144       ' display appropriate status image
145       If winner = GameStatus.PUSH Then ' push
146          statusPictureBox.Image =
147             Image.FromFile("blackjack_images/tie.png")
148       ElseIf winner = GameStatus.LOSE Then ' player loses
149          statusPictureBox.Image =
150             Image.FromFile("blackjack_images/lose.png")
151       ElseIf winner = GameStatus.BLACKJACK Then
152          ' player has blackjack
153          statusPictureBox.Image =
154             Image.FromFile("blackjack_images/blackjack.png")
155       Else ' player wins
156          statusPictureBox.Image =
157             Image.FromFile("blackjack_images/win.png")
158       End If
159
160       ' display final totals for dealer and player
161       dealerTotalLabel.Text =
162          "Dealer: " & dealer.GetHandValue(dealersCards)
163       playerTotalLabel.Text =
164          "Player: " & dealer.GetHandValue(playersCards)
165
166       ' reset controls for new game
167       stayButton.Enabled = False
168       hitButton.Enabled = False
169       dealButton.Enabled = True
170    End Sub ' GameOver
171
172    ' deal two cards each to dealer and player
173    Private Sub dealButton_Click(ByVal sender As System.Object,
174       ByVal e As System.EventArgs) Handles dealButton.Click
175       Dim card As String ' stores a card temporarily until added to a hand
176
177       ' clear card images
178       For Each cardImage As PictureBox In cardBoxes
179          cardImage.Image = Nothing
180       Next
181
182       statusPictureBox.Image = Nothing ' clear status image
```

Fig. 23.20 | Blackjack game that uses the `BlackjackService` web service. (Part 4 of 8.)

```
183         dealerTotalLabel.Text = String.Empty ' clear final total for dealer
184         playerTotalLabel.Text = String.Empty ' clear final total for player
185
186         ' create a new, shuffled deck on the web service host
187         dealer.Shuffle()
188
189         ' deal two cards to player
190         playersCards = dealer.DealCard() ' deal a card to player's hand
191
192         ' update GUI to display new card
193         DisplayCard(11, playersCards)
194         card = dealer.DealCard() ' deal a second card
195         DisplayCard(12, card) ' update GUI to display new card
196         playersCards &= vbTab & card ' add second card to player's hand
197
198         ' deal two cards to dealer, only display face of first card
199         dealersCards = dealer.DealCard() ' deal a card to dealer's hand
200         DisplayCard(0, dealersCards) ' update GUI to display new card
201         card = dealer.DealCard() ' deal a second card
202         DisplayCard(1, String.Empty) ' update GUI to show face-down card
203         dealersCards &= vbTab & card ' add second card to dealer's hand
204
205         stayButton.Enabled = True ' allow player to stay
206         hitButton.Enabled = True ' allow player to hit
207         dealButton.Enabled = False ' disable Deal Button
208
209         ' determine the value of the two hands
210         Dim dealerTotal As Integer = dealer.GetHandValue(dealersCards)
211         Dim playerTotal As Integer = dealer.GetHandValue(playersCards)
212
213         ' if hands equal 21, it is a push
214         If dealerTotal = playerTotal And dealerTotal = 21 Then
215            GameOver(GameStatus.PUSH)
216         ElseIf dealerTotal = 21 Then ' if dealer has 21, dealer wins
217            GameOver(GameStatus.LOSE)
218         ElseIf playerTotal = 21 Then ' player has blackjack
219            GameOver(GameStatus.BLACKJACK)
220         End If
221
222         currentDealerCard = 2 ' next dealer card has index 2 in cardBoxes
223         currentPlayerCard = 13 ' next player card has index 13 in cardBoxes
224      End Sub ' dealButton_Click
225
226      ' deal another card to player
227      Private Sub hitButton_Click(ByVal sender As System.Object,
228         ByVal e As System.EventArgs) Handles hitButton.Click
229         ' get player another card
230         Dim card As String = dealer.DealCard() ' deal new card
231         playersCards &= vbTab & card ' add new card to player's hand
232
233         ' update GUI to show new card
234         DisplayCard(currentPlayerCard, card)
235         currentPlayerCard += 1
```

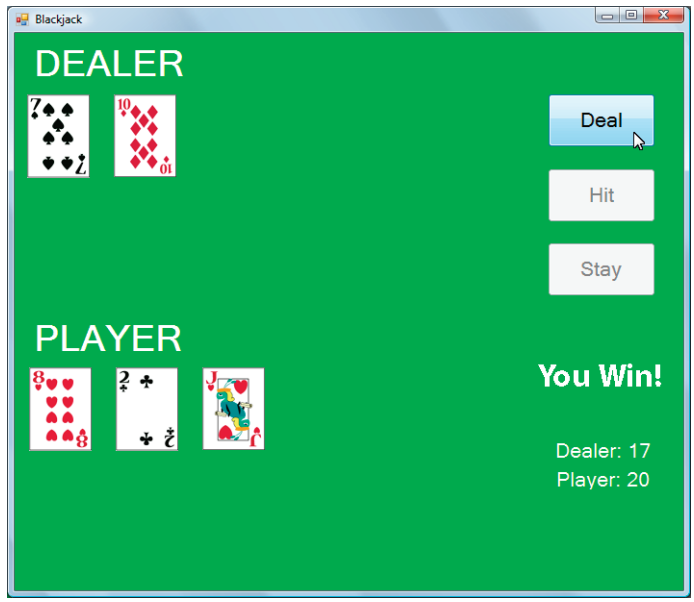
Fig. 23.20 | Blackjack game that uses the `BlackjackService` web service. (Part 5 of 8.)

```
236
237         ' determine the value of the player's hand
238         Dim total As Integer = dealer.GetHandValue(playersCards)
239
240         ' if player exceeds 21, house wins
241         If total > 21 Then
242            GameOver(GameStatus.LOSE)
243         End If
244
245         ' if player has 21,
246         ' they cannot take more cards, and dealer plays
247         If total = 21 Then
248            hitButton.Enabled = False
249            DealerPlay()
250         End If
251      End Sub ' hitButton_Click
252
253      ' play the dealer's hand after the play chooses to stay
254      Private Sub stayButton_Click(ByVal sender As System.Object,
255         ByVal e As System.EventArgs) Handles stayButton.Click
256         stayButton.Enabled = False ' disable Stay Button
257         hitButton.Enabled = False ' disable Hit Button
258         dealButton.Enabled = True ' re-enable Deal Button
259         DealerPlay() ' player chose to stay, so play the dealer's hand
260      End Sub ' stayButton_Click
261   End Class ' Blackjack
```

a) Initial cards dealt to the player and the dealer when the user presses the **Deal** button.

Fig. 23.20 | Blackjack game that uses the `BlackjackService` web service. (Part 6 of 8.)

b) Cards after the player presses the **Hit** button once, then the **Stay** button. In this case, the player wins the game with a higher total than the dealer.



c) Cards after the player presses the **Hit** button once, then the **Stay** button. In this case, the player busts (exceeds 21) and the dealer wins the game.

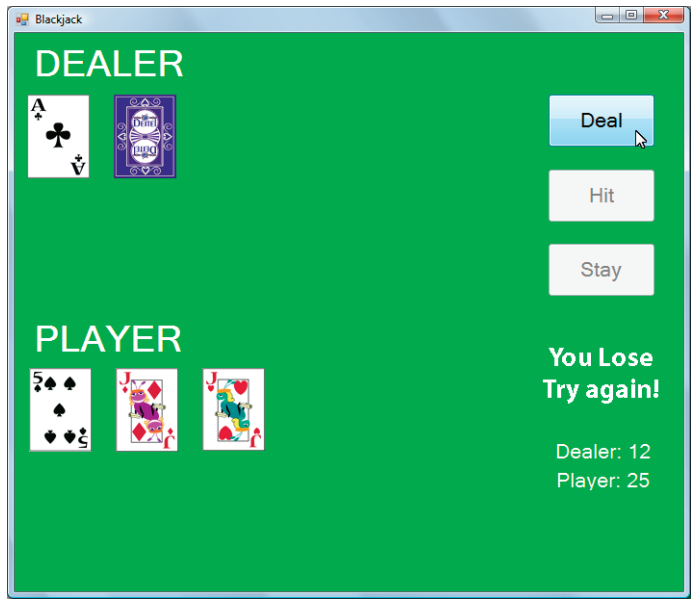
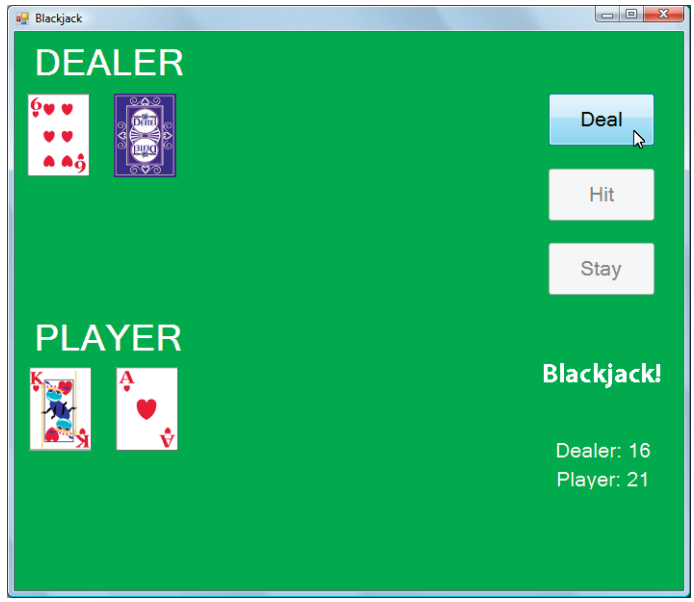


Fig. 23.20 | Blackjack game that uses the `BlackjackService` web service. (Part 7 of 8.)

d) Cards after the player presses the **Deal** button. In this case, the player wins with Blackjack because the first two cards are an ace and a card with a value of 10 (a jack in this case).



e) Cards after the player presses the **Stay** button. In this case, the player and dealer push—they have the same card total.

Fig. 23.20 | Blackjack game that uses the `BlackjackService` web service. (Part 8 of 8.)

Each player has 11 `PictureBoxes`—the maximum number of cards that can be dealt without exceeding 21 (that is, four aces, four twos and three threes). These `PictureBoxes` are placed in a `List` (lines 35–56), so we can index the `List` during the game to determine which `PictureBox` should display a particular card image. The images are located in the `blackjack_images` directory with this example. Drag this directory from Windows Explorer into your project. In the **Solution Explorer**, select all the files in that folder and set their **Copy to Output Directory** property to **Copy if newer**.

Method `GameOver` (lines 143–170) shows an appropriate message in the status `PictureBox` and displays the final point totals of both the dealer and the player. These values are obtained by calling the web service's `GetHandValue` method in lines 162 and 164. Method `GameOver` receives as an argument a member of the `GameStatus` enumeration (defined in lines 19–24). The enumeration represents whether the player tied, lost or won the game; its four members are `PUSH`, `LOSE`, `WIN` and `BLACKJACK`.

When the player clicks the **Deal** button, the event handler (lines 173–224) clears the `PictureBoxes` and the `Labels` displaying the final point totals. Line 187 shuffles the deck by calling the web service's `Shuffle` method, then the player and dealer receive two cards each (returned by calls to the web service's `DealCard` method in lines 190, 194, 199 and 201). Lines 210–211 evaluate both the dealer's and player's hands by calling the web service's `GetHandValue` method. If the player and the dealer both obtain scores of 21, the program calls method `GameOver`, passing `GameStatus.PUSH`. If only the player has 21 after the first two cards are dealt, the program passes `GameStatus.BLACKJACK` to method `GameOver`. If only the dealer has 21, the program passes `GameStatus.LOSE` to method `GameOver`.

If `dealButton_Click` does not call `GameOver`, the player can take more cards by clicking the **Hit** button. The event handler for this button is in lines 227–251. Each time a player clicks **Hit**, the program deals the player one more card (line 230), displaying it in the GUI. Line 238 evaluates the player's hand. If the player exceeds 21, the game is over, and the player loses. If the player has exactly 21, the player cannot take any more cards, and method `DealerPlay` (lines 61–98) is called, causing the dealer to keep taking cards until the dealer's hand has a value of 17 or more (lines 71–79). If the dealer exceeds 21, the player wins (line 86); otherwise, the values of the hands are compared, and `GameOver` is called with the appropriate argument (lines 90–96).

Clicking the **Stay** button indicates that a player does not want to be dealt another card. The event handler for this button (lines 254–260) disables the **Hit** and **Stay** buttons, then calls method `DealerPlay`.

Method `DisplayCard` (lines 101–139) updates the GUI to display a newly dealt card. The method takes as arguments an integer representing the index of the `PictureBox` in the `List` that must have its image set, and a `String` representing the card. An empty `String` indicates that we wish to display the card face down. If method `DisplayCard` receives a `String` that's not empty, the program extracts the face and suit from the `String` and uses this information to find the correct image. The `Select Case` statement (lines 125–134) converts the number representing the suit to an `Integer` and assigns the appropriate character literal to `suitLetter` (`c` for clubs, `d` for diamonds, `h` for hearts and `s` for spades). The character in `suitLetter` is used to complete the image's file name (lines 137–138).

23.10 Airline Reservation Web Service: Database Access and Invoking a Service from ASP.NET

Our prior examples accessed web services from Windows Forms applications. You can just as easily use web services in ASP.NET web applications. In fact, because web-based businesses are becoming increasingly prevalent, it is common for web applications to consume web services. Figures 23.21 and 23.22 present the interface and class, respectively, for an airline reservation service that receives information regarding the type of seat a customer wishes to reserve, checks a database to see if such a seat is available and, if so, makes a reservation. Later in this section, we present an ASP.NET web application that allows a customer to specify a reservation request, then uses the airline reservation web service to attempt to execute the request. The code and database used in this example are provided with the chapter's examples, which can be downloaded from `www.deitel.com/books/vb2010htp`.

```
1  ' Fig. 23.21: IReservationService.vb
2  ' Airline reservation WCF web-service interface.
3  <ServiceContract()>
4  Public Interface IReservationService
5     ' reserves a seat
6     <OperationContract()>
7     Function Reserve(ByVal seatType As String,
8        ByVal classType As String) As Boolean
9  End Interface ' IReservationService
```

Fig. 23.21 | Airline reservation WCF web-service interface.

```
1  ' Fig. 23.22: ReservationService.vb
2  ' Airline reservation WCF web service.
3  Public Class ReservationService
4     Implements IReservationService
5
6     ' create ticketsDB object to access Tickets database
7     Private ticketsDB As New TicketsDataContext()
8
9     ' checks database to determine whether matching seat is available
10    Public Function Reserve(ByVal seatType As String,
11       ByVal classType As String) As Boolean _
12       Implements IReservationService.Reserve
13
14       ' LINQ query to find seats matching the parameters
15       Dim result =
16          From seat In ticketsDB.Seats
17          Where (seat.Taken = 0) And (seat.SeatType = seatType)
18             And (seat.SeatClass = classType)
19
20       ' if the number of seats returned is nonzero,
21       ' obtain the first matching seat number and mark it as taken
22       If result.Count() <> 0 Then
```

Fig. 23.22 | Airline reservation WCF web service. (Part 1 of 2.)

```
            ' get first available seat
            Dim firstAvailableSeat As Seat = result.First()
            firstAvailableSeat.Taken = 1 ' mark the seat as taken
            ticketsDB.SubmitChanges() ' update
            Return True ' seat was reserved
         End If

         Return False ' no seat was reserved
      End Function ' Reserve
   End Class ' ReservationService
```

Fig. 23.22 | Airline reservation WCF web service. (Part 2 of 2.)

In Chapter 12, you learned how to use LINQ to SQL to extract data from a database. We added the `Tickets.mdf` database and corresponding LINQ to SQL classes to create a `DataContext` object (line 7) for our ticket reservation system. `Tickets.mdf` database contains the `Seats` table with four columns—the seat number (1–10), the seat type (`Window`, `Middle` or `Aisle`), the class type (`Economy` or `First`) and a column containing either `1` (true) or `0` (false) to indicate whether the seat is taken.

This web service has a single method—`Reserve` (lines 10–31)—which searches a seat database (`Tickets.mdf`) to locate a seat matching a user's request. If it finds an appropriate seat, `Reserve` updates the database, makes the reservation and returns `True`; otherwise, no reservation is made, and the method returns `False`. The statements in lines 15–18 and lines 22–28, which query and update the database, use LINQ to SQL.

`Reserve` receives two parameters—a `String` representing the seat type (that is, `Window`, `Middle` or `Aisle`) and a `String` representing the class type (that is, `Economy` or `First`). Our database contains four columns—the seat number (that is, `1`–`10`), the seat type (that is, `Window`, `Middle` or `Aisle`), the class type (that is, `Economy` or `First`) and a column containing either `1` (true) or `0` (false) to indicate whether the seat is taken. Lines 16–18 retrieve the seat numbers of any available seats matching the requested seat and class type with the results of a query. In line 22, if the number of results in the query is not zero, there was at least one seat that matched the user's request. In this case, the web service reserves the first matching seat. We obtain the seat in line 24 by accessing the query's first result. Line 25 marks the seat as taken and line 26 submits the changes to the database. Method `Reserve` returns `True` (line 27) to indicate that the reservation was successful. If there are no matching seats, `Reserve` returns `False` (line 30) to indicate that no seats matched the user's request.

Creating a Web Form to Interact with the Airline Reservation Web Service

Figure 23.23 presents an ASP.NET page through which users can select seat types. This page allows users to reserve a seat on the basis of its class (`Economy` or `First`) and location (`Aisle`, `Middle` or `Window`) in a row of seats. The page then uses the airline reservation web service to carry out user requests. If the database request is not successful, the user is instructed to modify the request and try again. When you create this ASP.NET application, remember to add a service reference to the `ReservationService`.

This page defines two `DropDownList` objects and a `Button`. One `DropDownList` displays all the seat types from which users can select (`Aisle`, `Middle`, `Window`). The second provides choices for the class type. Users click the `Button` named `reserveButton` to

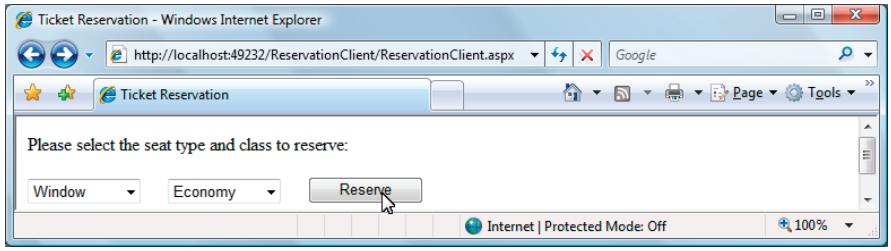


Fig. 23.23 | ASPX file that takes reservation information.

submit requests after making selections from the `DropDownLists`. The page also defines an initially blank `Label` named `errorLabel`, which displays an appropriate message if no seat matching the user's selection is available. Line 9 of the code-behind file (Fig. 23.24) attaches an event handler to `reserveButton`.

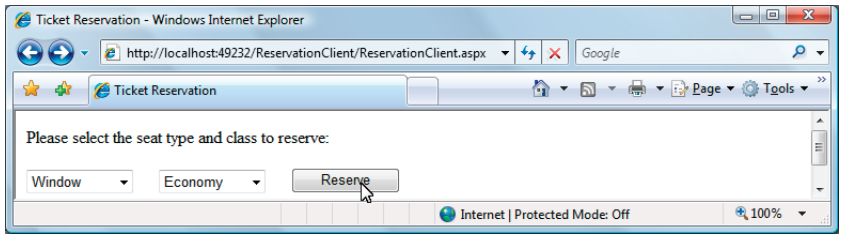
Line 6 of Fig. 23.24 creates a `ReservationServiceClient` proxy object. When the user clicks **Reserve** (Fig. 23.25), the `reserveButton_Click` event handler (lines 8–29 of Fig. 23.24) executes, and the page reloads. The event handler calls the web service's

```
1  ' Fig. 23.24: ReservationClient.aspx.vb
2  ' ReservationClient code-behind file.
3  Partial Class ReservationClient
4     Inherits System.Web.UI.Page
5     ' object of proxy type used to connect to Reservation service
6     Private ticketAgent As New ServiceReference.ReservationServiceClient()
7
8     Protected Sub reserveButton_Click(ByVal sender As Object,
9        ByVal e As System.EventArgs) Handles reserveButton.Click
10
11       ' if the ticket is reserved
12       If ticketAgent.Reserve(seatList.SelectedItem.Text,
13          classList.SelectedItem.Text.ToString()) Then
14
15          ' hide other controls
16          instructionsLabel.Visible = False
17          seatList.Visible = False
18          classList.Visible = False
19          reserveButton.Visible = False
20          errorLabel.Visible = False
21
22          ' display message indicating success
23          Response.Write("Your reservation has been made. Thank you.")
24       Else ' service method returned false, so signal failure
25          ' display message in the initially blank errorLabel
26          errorLabel.Text = "This type of seat is not available. " &
27             "Please modify your request and try again."
28       End If
29    End Sub ' reserveButton_Click
30 End Class ' ReservationClient
```

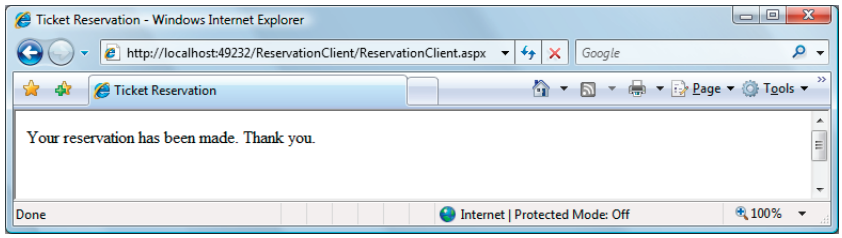
Fig. 23.24 | `ReservationClient` code-behind file.

`Reserve` method and passes to it the selected seat and class type as arguments (lines 12–13). If `Reserve` returns `True`, the application hides the GUI controls and displays a message thanking the user for making a reservation (line 23); otherwise, the application notifies the user that the type of seat requested is not available and instructs the user to try again (lines 26–27). You can use the techniques presented in Chapter 13 to build this ASP.NET Web Form. Figure 23.25 shows several user interactions with this web application.

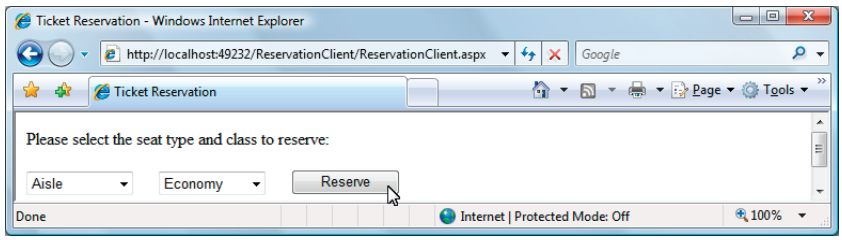
a) Selecting a seat.



b) Seat is reserved successfully.



c) Attempting to reserve another seat.



d) No seats match the requested type and class.

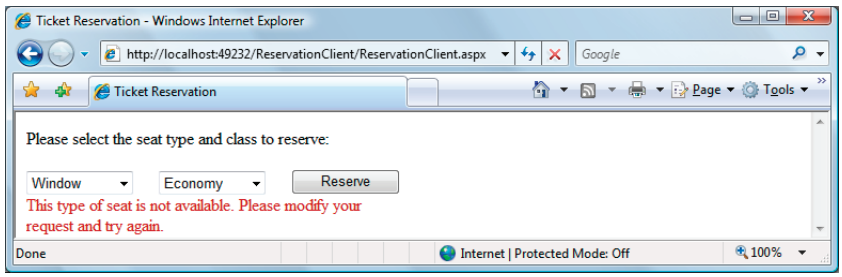


Fig. 23.25 | Ticket reservation web-application sample execution.

23.11 Equation Generator: Returning User-Defined Types

With the exception of the `WelcomeRESTJSONService` (Fig. 23.15), the web services we've demonstrated all received and returned primitive-type instances. It is also possible to process instances of complete user-defined types in a web service. These types can be passed to or returned from web-service methods.

This section presents an `EquationGenerator` web service that generates random arithmetic equations of type `Equation`. The client is a math-tutoring application that inputs information about the mathematical question that the user wishes to attempt (addition, subtraction or multiplication) and the skill level of the user (1 specifies equations using numbers from 1 to 10, 2 specifies equations involving numbers from 10 to 100, and 3 specifies equations containing numbers from 100 to 1000). The web service then generates an equation consisting of random numbers in the proper range. The client application receives the `Equation` and displays the sample question to the user.

Defining Class `Equation`

We define class `Equation` in Fig. 23.26. Lines 16–33 define a constructor that takes three arguments—two `Integer`s representing the left and right operands and a `String` that represents the arithmetic operation to perform. The constructor sets the `leftOperand`, `rightOperand` and `operationType` instance variables, then calculates the appropriate result. The parameterless constructor (lines 11–13) calls the three-argument constructor (lines 16–33) and passes default values.

```
1  ' Fig. 23.26: Equation.vb
2  ' Class Equation that contains information about an equation.
3  <DataContract()>
4  Public Class Equation
5     Private leftOperand As Integer ' number to the left of the operator
6     Private rightOperand As Integer ' number to the right of the operator
7     Private resultValue As Integer ' result of the operation
8     Private operationType As String ' type of the operation
9
10    ' required default constructor
11    Public Sub New()
12       MyClass.New(0, 0, "add")
13    End Sub ' parameterless New
14
15    ' three-argument constructor for class Equation
16    Public Sub New(ByVal leftValue As Integer,
17       ByVal rightValue As Integer, ByVal type As String)
18
19       Left = leftValue
20       Right = rightValue
21
22       Select Case type ' perform appropriate operation
23          Case "add" ' addition
24             Result = leftOperand + rightOperand
25             operationType = "+"
```

Fig. 23.26 | Class `Equation` that contains information about an equation. (Part 1 of 3.)

```
            Case "subtract" ' subtraction
               Result = leftOperand - rightOperand
               operationType = "-"
            Case "multiply" ' multiplication
               Result = leftOperand * rightOperand
               operationType = "*"
         End Select
      End Sub ' three-parameter New

      ' return string representation of the Equation object
      Public Overrides Function ToString() As String
         Return leftOperand.ToString() & " " & operationType & " " &
            rightOperand.ToString() & " = " & resultValue.ToString()
      End Function ' ToString

      ' property that returns a string representing left-hand side
      <DataMember()>
      Public Property LeftHandSide() As String
         Get
            Return leftOperand.ToString() & " " & operationType & " " &
               rightOperand.ToString()
         End Get

         Set(ByVal value As String) ' required set accessor
            ' empty body
         End Set
      End Property ' LeftHandSide

      ' property that returns a string representing right-hand side
      <DataMember()>
      Public Property RightHandSide() As String
         Get
            Return resultValue.ToString()
         End Get

         Set(ByVal value As String) ' required set accessor
            ' empty body
         End Set
      End Property ' RightHandSide

      ' property to access the left operand
      <DataMember()>
      Public Property Left() As Integer
         Get
            Return leftOperand
         End Get

         Set(ByVal value As Integer)
            leftOperand = value
         End Set
      End Property ' Left

```

Fig. 23.26 | Class `Equation` that contains information about an equation. (Part 2 of 3.)

```
78       ' property to access the right operand
79       <DataMember()>
80       Public Property Right() As Integer
81          Get
82             Return rightOperand
83          End Get
84
85          Set(ByVal value As Integer)
86             rightOperand = value
87          End Set
88       End Property ' Right
89
90       ' property to access the result of applying
91       ' an operation to the left and right operands
92       <DataMember()>
93       Public Property Result() As Integer
94          Get
95             Return resultValue
96          End Get
97
98          Set(ByVal value As Integer)
99             resultValue = value
100         End Set
101      End Property ' Result
102
103      ' property to access the operation
104      <DataMember()>
105      Public Property Operation() As String
106         Get
107            Return operationType
108         End Get
109
110         Set(ByVal value As String)
111            operationType = value
112         End Set
113      End Property ' Operation
114   End Class ' Equation
```

Fig. 23.26 | Class `Equation` that contains information about an equation. (Part 3 of 3.)

Class `Equation` defines properties `LeftHandSide` (lines 43–52), `RightHandSide` (lines 56–64), `Left` (lines 68–76), `Right` (lines 80–88), `Result` (lines 93–101) and `Operation` (lines 105–113). The web service client does not need to modify the values of properties `LeftHandSide` and `RightHandSide`. However, recall that a property can be serialized only if it has both a `Get` and a `Set` accessor—this is true even if the `Set` accessor has an empty body. Each of the properties is preceded by the `DataMember` attribute to indicate that it should be serialized. `LeftHandSide` (lines 43–52) returns a `String` representing everything to the left of the equals (=) sign in the equation, and `RightHandSide` (lines 56–64) returns a `String` representing everything to the right of the equals (=) sign. `Left` (lines 68–76) returns the `Integer` to the left of the operator (known as the left operand), and `Right` (lines 80–88) returns the `Integer` to the right of the operator (known as the right operand). `Result` (lines 93–101) returns the solution to the equation, and `Operation`

(lines 105–113) returns the operator in the equation. The client in this case study does not use the `RightHandSide` property, but we included it in case future clients choose to use it. Method `ToString` (lines 36–39) returns a `String` representation of the equation.

23.11.1 Creating the REST-Based XML EquationGenerator Web Service

Figures 23.27 and 23.28 present the interface and class for the `EquationGeneratorService` web service, which creates random, customized `Equations`. This web service contains only method `GenerateEquation` (lines 7–27 of Fig. 23.28), which takes two parameters—a `String` representing the mathematical operation (`"add"`, `"subtract"` or `"multiply"`) and a `String` representing the difficulty level. When line 26 of Fig. 23.28 returns the `Equation`, it is serialized as XML by default and sent to the client. We'll do this with JSON as well in Section 23.11.3. Recall from Section 23.7.2 that you must modify the `Web.config` file to enable REST support as well.

```
1  ' Fig. 23.27: IEquationGeneratorService.vb
2  ' WCF REST service interface to create random equations based on a
3  ' specified operation and difficulty level.
4  Imports System.ServiceModel.Web
5
6  <ServiceContract()>
7  Public Interface IEquationGeneratorService
8     ' method to generate a math equation
9     <OperationContract()>
10    <WebGet(UriTemplate:="equation/{operation}/{level}")>
11    Function GenerateEquation(ByVal operation As String,
12       ByVal level As String) As Equation
13 End Interface ' IEquationGeneratorService
```

Fig. 23.27 | WCF REST service interface to create random equations based on a specified operation and difficulty level.

```
1  ' Fig. 23.28: EquationGeneratorService.vb
2  ' WCF REST service to create random equations based on a
3  ' specified operation and difficulty level.
4  Public Class EquationGeneratorService
5     Implements IEquationGeneratorService
6     ' method to generate a math equation
7     Public Function GenerateEquation(ByVal operation As String,
8        ByVal level As String) As Equation _
9        Implements IEquationGeneratorService.GenerateEquation
10
11       ' convert level from String to Integer
12       Dim digits = Convert.ToInt32(level)
13
14       ' calculate maximum and minimum number to be used
15       Dim maximum As Integer = Convert.ToInt32(Math.Pow(10, digits))
16       Dim minimum As Integer = Convert.ToInt32(Math.Pow(10, digits - 1))
```

Fig. 23.28 | WCF REST service to create random equations based on a specified operation and difficulty level. (Part 1 of 2.)

```
17
18          Dim randomObject As New Random() ' used to generate random numbers
19
20          ' create Equation consisting of two random
21          ' numbers in the range minimum to maximum
22          Dim newEquation As New Equation(
23             randomObject.Next(minimum, maximum),
24             randomObject.Next(minimum, maximum), operation)
25
26          Return newEquation
27       End Function ' GenerateEquation
28    End Class ' EquationGeneratorService
```

Fig. 23.28 | WCF REST service to create random equations based on a specified operation and difficulty level. (Part 2 of 2.)

23.11.2 Consuming the REST-Based XML EquationGenerator Web Service

The `MathTutor` application (Fig. 23.29) calls the `EquationGenerator` web service's `GenerateEquation` method to create an `Equation` object. The tutor then displays the left-hand side of the `Equation` and waits for user input.

The default setting for the difficulty level is `1`, but the user can change this by choosing a level from the `RadioButtons` in the `GroupBox` labeled **Difficulty**. Clicking any of the levels invokes the corresponding `RadioButton`'s `CheckedChanged` event handler (lines 85–103), which sets integer `level` to the level selected by the user. Although the default setting for the question type is **Addition**, the user also can change this by selecting one of the `RadioButtons` in the `GroupBox` labeled **Operation**. Doing so invokes the corresponding operation's event handlers in lines 64–82, which assigns to `String operation` the symbol corresponding to the user's selection. Each event handler also updates the `Text` property of the **Generate** button to match the newly selected operation.

Line 13 defines the `WebClient` that is used to invoke the web service. Event handler `generateButton_Click` (lines 16–23) invokes `EquationGeneratorService` method `GenerateEquation` (line 20–22) asynchronously using the web service's `UriTemplate` specified at line 10 in Fig. 23.27. When the response arrives, the `DownloadStringCompleted` event handler (lines 26–41) parses the XML response (line 33), uses XML Axis properties to obtain the left side of the equation (line 34) and stores the result (line 35). Then, the handler displays the left-hand side of the equation in `questionLabel` (line 37) and enables `okButton` so that the user can enter an answer. When the user clicks **OK**, `okButton_Click` (lines 44–61) checks whether the user provided the correct answer.

```
1  ' Fig. 23.29: MathTutor.vb
2  ' Math tutor using EquationGeneratorService to create equations.
3  Imports System.Net
4  Imports System.Xml.Linq
5  Imports <xmlns="http://schemas.datacontract.org/2004/07/">
```

Fig. 23.29 | Math tutor using XML version of `EquationGeneratorService` to create equations. (Part 1 of 4.)

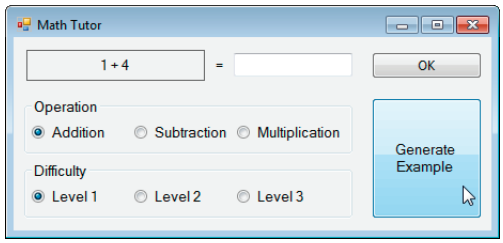
```
6
7  Public Class MathTutor
8     Private operation As String = "add" ' the default operation
9     Private level As Integer = 1 ' the default difficulty level
10    Private leftHandSide As String ' the left side of the equation
11    Private result As Integer ' the answer
12
13    Private WithEvents service As New WebClient() ' used to invoke service
14
15    ' generates a new equation when user clicks button
16    Private Sub generateButton_Click(ByVal sender As System.Object,
17       ByVal e As System.EventArgs) Handles generateButton.Click
18
19       ' send request to EquationGeneratorServiceXML
20       service.DownloadStringAsync(New Uri(
21          "http://localhost:49593/EquationGeneratorServiceXML/" &
22          "Service.svc/equation/" & operation & "/" & level))
23    End Sub ' generateButton_Click
24
25    ' process web-service response
26    Private Sub service_DownloadStringCompleted(ByVal sender As Object,
27       ByVal e As System.Net.DownloadStringCompletedEventArgs) _
28       Handles service.DownloadStringCompleted
29
30       ' check if any errors occurred in retrieving service data
31       If e.Error Is Nothing Then
32          ' parse response and get LeftHandSide and Result values
33          Dim xmlResponse = XDocument.Parse(e.Result)
34          leftHandSide = xmlResponse.<Equation>.<LeftHandSide>.Value
35          result = Convert.ToInt32(xmlResponse.<Equation>.<Result>.Value)
36
37          questionLabel.Text = leftHandSide ' display left side of equation
38          okButton.Enabled = True ' enable okButton
39          answerTextBox.Enabled = True ' enable answerTextBox
40       End If
41    End Sub ' service_DownloadStringCompleted
42
43    ' check user's answer
44    Private Sub okButton_Click(ByVal sender As System.Object,
45       ByVal e As System.EventArgs) Handles okButton.Click
46
47       If Not String.IsNullOrEmpty(answerTextBox.Text) Then
48          ' get user's answer
49          Dim userAnswer As Integer = Convert.ToInt32(answerTextBox.Text)
50
51          ' determine whether user's answer is correct
52          If result = userAnswer Then
53             questionLabel.Text = String.Empty ' clear question
54             answerTextBox.Clear() ' clear answer
55             okButton.Enabled = False ' disable OK button
56             MessageBox.Show("Correct! Good job!")
```

Fig. 23.29 | Math tutor using XML version of `EquationGeneratorService` to create equations. (Part 2 of 4.)

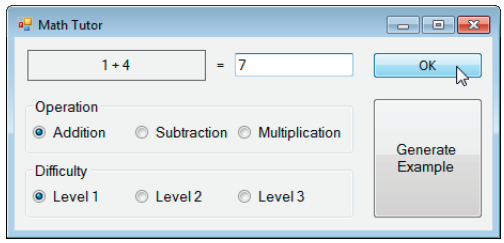
```
57            Else
58               MessageBox.Show("Incorrect. Try again.")
59            End If
60         End If
61      End Sub ' okButton_Click
62
63      ' set the operation to addition
64      Private Sub additionRadioButton_CheckedChanged(
65         ByVal sender As System.Object, ByVal e As System.EventArgs) _
66         Handles additionRadioButton.CheckedChanged
67         operation = "add"
68      End Sub ' additionRadioButton_CheckedChanged
69
70      ' set the operation to subtraction
71      Private Sub subtractionRadioButton_CheckedChanged(
72         ByVal sender As System.Object, ByVal e As System.EventArgs) _
73         Handles subtractionRadioButton.CheckedChanged
74         operation = "subtract"
75      End Sub ' subtractionRadioButton_CheckedChanged
76
77      ' set the operation to multiplication
78      Private Sub multiplicationRadioButton_CheckedChanged(
79         ByVal sender As System.Object, ByVal e As System.EventArgs) _
80         Handles multiplicationRadioButton.CheckedChanged
81         operation = "multiply"
82      End Sub ' multiplicationRadioButton_CheckedChanged
83
84      ' set difficulty level to 1
85      Private Sub levelOneRadioButton_CheckedChanged(
86         ByVal sender As System.Object, ByVal e As System.EventArgs) _
87         Handles levelOneRadioButton.CheckedChanged
88         level = 1
89      End Sub ' levelOneRadioButton_CheckedChanged
90
91      ' set difficulty level to 2
92      Private Sub levelTwoRadioButton_CheckedChanged(
93         ByVal sender As System.Object, ByVal e As System.EventArgs) _
94         Handles levelTwoRadioButton.CheckedChanged
95         level = 2
96      End Sub ' levelTwoRadioButton_CheckedChanged
97
98      ' set difficulty level to 3
99      Private Sub levelThreeRadioButton_CheckedChanged(
100        ByVal sender As System.Object, ByVal e As System.EventArgs) _
101        Handles levelThreeRadioButton.CheckedChanged
102        level = 3
103     End Sub ' levelThreeRadioButton_CheckedChanged
104  End Class ' MathTutor
```

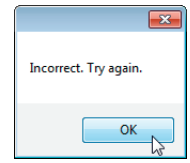
Fig. 23.29 | Math tutor using XML version of `EquationGeneratorService` to create equations. (Part 3 of 4.)

a) Generating a level 1 addition equation.

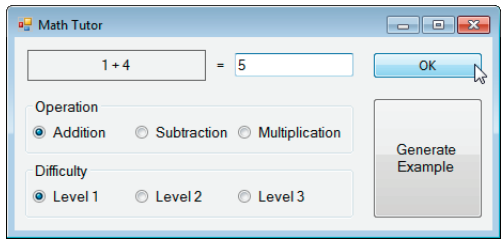


b) Answering the question incorrectly.





c) Answering the question correctly.



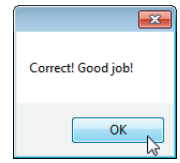


Fig. 23.29 | Math tutor using XML version of `EquationGeneratorService` to create equations. (Part 4 of 4.)

23.11.3 Creating the REST-Based JSON WCF `EquationGenerator` Web Service

You can set the web service to return JSON data instead of XML. Figure 23.30 is a modified `IEquationGeneratorService` interface for a service that returns an `Equation` in JSON format. The `ResponseFormat` property (line 10) is added to the `WebGet` attribute and set to `WebMessageFormat.Json`. We don't show the implementation of this interface here, because it is identical to that of Fig. 23.28. This shows how flexible WCF can be.

```
1  ' Fig. 23.30: IEquationGeneratorService.vb
2  ' WCF REST service interface to create random equations based on a
3  ' specified operation and difficulty level.
4  Imports System.ServiceModel.Web
5
6  <ServiceContract()>
7  Public Interface IEquationGeneratorService
8     ' method to generate a math equation
9     <OperationContract()>
10    <WebGet(ResponseFormat:=WebMessageFormat.Json,
11       UriTemplate:="equation/{operation}/{level}")>
12    Function GenerateEquation(ByVal operation As String,
13       ByVal level As String) As Equation
14 End Interface ' IEquationGeneratorService
```

Fig. 23.30 | WCF REST service interface to create random equations based on a specified operation and difficulty level.

23.11.4 Consuming the REST-Based JSON WCF EquationGenerator Web Service

A modified `MathTutor` application (Fig. 23.31) accesses the URI of the `EquationGenerator` web service to get the JSON object (lines 19–21). We define a JSON representation of an `Equation` object for the serializer in Fig. 23.32. The JSON object is deserialized using a `DataContractJsonSerializer` (lines 32–35) and converted into an `Equation` object. We use the `LeftHandSide` field of the deserialized object (line 38) to display the left side of the equation and the `Result` field (line 51–52) to obtain the answer.

```
1  ' Fig. 23.31: MathTutor.vb
2  ' Math tutor using EquationGeneratorServiceJSON to create equations.
3  Imports System.Net
4  Imports System.IO
5  Imports System.Text
6  Imports System.Runtime.Serialization.Json
7
8  Public Class MathTutor
9     Private operation As String = "add" ' the default operation
10    Private level As Integer = 1 ' the default difficulty level
11    Private currentEquation As Equation ' represents the Equation
12    Private WithEvents service As New WebClient() ' used to invoke service
13
14    ' generates a new equation when user clicks button
15    Private Sub generateButton_Click(ByVal sender As System.Object,
16       ByVal e As System.EventArgs) Handles generateButton.Click
17
18       ' send request to EquationGeneratorServiceJSON
19       service.DownloadStringAsync(New Uri(
20          "http://localhost:49817/EquationGeneratorServiceJSON/" &
21          "Service.svc/equation/" & operation & "/" & level))
22    End Sub ' generateButton_Click
```

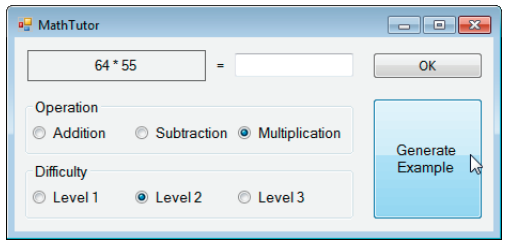
Fig. 23.31 | Math tutor using JSON version of `EquationGeneratorServiceJSON`. (Part 1 of 4.)

```
23
24      ' process web-service response
25      Private Sub service_DownloadStringCompleted(ByVal sender As Object,
26         ByVal e As System.Net.DownloadStringCompletedEventArgs) _
27         Handles service.DownloadStringCompleted
28
29         ' check if any errors occurred in retrieving service data
30         If e.Error Is Nothing Then
31            ' deserialize response into an equation object
32            Dim JSONSerializer As New
33               DataContractJsonSerializer(GetType(Equation))
34            currentEquation = CType(JSONSerializer.ReadObject(New
35               MemoryStream(Encoding.Unicode.GetBytes(e.Result))), Equation)
36
37            ' display left side of equation
38            questionLabel.Text = currentEquation.LeftHandSide
39            okButton.Enabled = True ' enable okButton
40            answerTextBox.Enabled = True ' enable answerTextBox
41         End If
42      End Sub ' service_DownloadStringCompleted
43
44      ' check user's answer
45      Private Sub okButton_Click(ByVal sender As System.Object,
46         ByVal e As System.EventArgs) Handles okButton.Click
47
48         ' check if answer field is filled
49         If Not String.IsNullOrEmpty(answerTextBox.Text) Then
50            ' determine whether user's answer is correct
51            If currentEquation.Result =
52               Convert.ToInt32(answerTextBox.Text) Then
53
54               questionLabel.Text = String.Empty ' clear question
55               answerTextBox.Clear() ' clear answer
56               okButton.Enabled = False ' disable OK button
57               MessageBox.Show("Correct! Good job!")
58            Else
59               MessageBox.Show("Incorrect. Try again.")
60            End If
61         End If
62      End Sub ' okButton_Click
63
64      ' set the operation to addition
65      Private Sub additionRadioButton_CheckedChanged(
66         ByVal sender As System.Object, ByVal e As System.EventArgs) _
67         Handles additionRadioButton.CheckedChanged
68         operation = "add"
69      End Sub ' additionRadioButton_CheckedChanged
70
71      ' set the operation to subtraction
72      Private Sub subtractionRadioButton_CheckedChanged(
73         ByVal sender As System.Object, ByVal e As System.EventArgs) _
74         Handles subtractionRadioButton.CheckedChanged
```

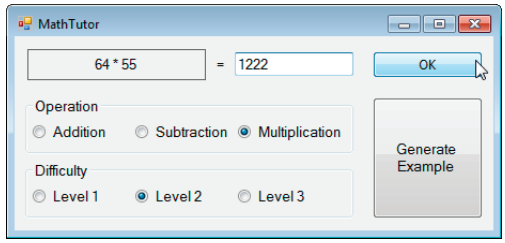
Fig. 23.31 | Math tutor using JSON version of `EquationGeneratorServiceJSON`. (Part 2 of 4.)

```
         operation = "subtract"
      End Sub ' subtractionRadioButton_CheckedChanged

      ' set the operation to multiplication
      Private Sub multiplicationRadioButton_CheckedChanged(
         ByVal sender As System.Object, ByVal e As System.EventArgs) _
         Handles multiplicationRadioButton.CheckedChanged
         operation = "multiply"
      End Sub ' multiplicationRadioButton_CheckedChanged

      ' set difficulty level to 1
      Private Sub levelOneRadioButton_CheckedChanged(
         ByVal sender As System.Object, ByVal e As System.EventArgs) _
         Handles levelOneRadioButton.CheckedChanged
         level = 1
      End Sub ' levelOneRadioButton_CheckedChanged

      ' set difficulty level to 2
      Private Sub levelTwoRadioButton_CheckedChanged(
         ByVal sender As System.Object, ByVal e As System.EventArgs) _
         Handles levelTwoRadioButton.CheckedChanged
         level = 2
      End Sub ' levelTwoRadioButton_CheckedChanged

      ' set difficulty level to 3
      Private Sub levelThreeRadioButton_CheckedChanged(
         ByVal sender As System.Object, ByVal e As System.EventArgs) _
         Handles levelThreeRadioButton.CheckedChanged
         level = 3
      End Sub ' levelThreeRadioButton_CheckedChanged
   End Class ' MathTutor
```

a) Generating a level 2 multiplication equation.



b) Answering the question incorrectly.



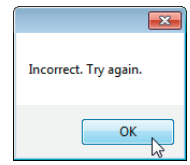


Fig. 23.31 | Math tutor using JSON version of `EquationGeneratorServiceJSON`. (Part 3 of 4.)

c) Answering the question correctly.

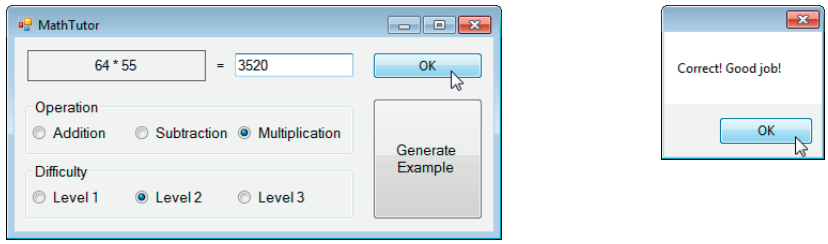


Fig. 23.31 | Math tutor using JSON version of `EquationGeneratorServiceJSON`. (Part 4 of 4.)

```
1  ' Fig. 23.32: Equation.vb
2  ' Equation class representing a JSON object.
3  <Serializable()>
4  Public Class Equation
5     Public Left As Integer
6     Public LeftHandSide As String
7     Public Operation As String
8     Public Result As Integer
9     Public Right As Integer
10    Public RightHandSide As String
11 End Class ' Equation
```

Fig. 23.32 | `Equation` class representing a JSON object.

23.12 Wrap-Up

This chapter introduced WCF web services—a set of technologies for building distributed systems in which system components communicate with one another over networks. You learned that a web service is a class that allows client software to call the web service's methods remotely via common data formats and protocols, such as XML, JSON, HTTP, SOAP and REST. We also discussed several benefits of distributed computing with web services.

We discussed how Visual Basic 2010 Express, Visual Web Developer 2010 Express, and WCF facilitate publishing and consuming web services. You learned how to define web services and methods using both SOAP protocol and REST architecture, and how to return data in both XML and JSON formats. You consumed SOAP-based web services using proxy classes to call the web service's methods. You also consumed REST-based web services using class `WebClient`. We built both Windows applications and ASP.NET web applications as web-service clients. After explaining the mechanics of web services through our `Welcome` examples, we demonstrated more sophisticated web services that use session tracking, database access and user-defined types.

23.13 Deitel Web Services Resource Centers

To learn more about web services, check out our web services Resource Centers at:

```
www.deitel.com/WebServices/
www.deitel.com/RESTWebServices/
```

You'll find articles, samples chapters and tutorials that discuss XML, web-services specifications, SOAP, WSDL, UDDI, .NET web services, consuming XML web services, and web-services architecture. You'll learn how to build your own Yahoo! maps mashup and applications that work with the Yahoo! Music Engine. You'll find information about Amazon's web services including the Amazon E-Commerce Service (ECS), Amazon historical pricing, Amazon Mechanical Turk, Amazon S3 (Simple Storage Service) and the Scalable Simple Queue Service (SQS). You'll learn how to use web services from several other companies including eBay, Google and Microsoft. You'll find REST web services best practices and guidelines. You'll also learn how to use REST web services with other technologies including SOAP, Rails, Windows Communication Foundation (`WCF`) and more. You can view the complete list of Deitel Resource Centers at `www.deitel.com/ResourceCenters.html`.

Summary

Section 23.1 Introduction

- WCF is a set of technologies for building distributed systems in which system components communicate with one another over networks. WCF uses a common framework for all communication between systems, so you need to learn only one programming model to use WCF.
- WCF web services promote software reusability in distributed systems that typically execute across the Internet.
- Simple Object Access Protocol (SOAP) is an XML-based protocol describing how to mark up requests and responses so that they can be sent via protocols such as HTTP. SOAP uses a standardized XML-based format to enclose data in a message.
- Representational State Transfer (REST) is a network architecture that uses the web's traditional request/response mechanisms such as `GET` and `POST` requests. REST-based systems do not require data to be wrapped in a special message format.

Section 23.2 WCF Services Basics

- WCF service has three key components—addresses, bindings and contracts.
- An address represents the service's location or endpoint, which includes the protocol and network address used to access the service.
- A binding specifies how a client communicates with the service, such as through SOAP protocol or REST architecture. Bindings can also specify other options, such as security constraints.
- A contract is an interface representing the service's methods and their return types. The service's contract allows clients to interact with the service.
- The machine on which the web service resides is referred to as a web service host.

Section 23.3 Simple Object Access Protocol (SOAP)

- The Simple Object Access Protocol (SOAP) is a platform-independent protocol that uses XML to make remote procedure calls, typically over HTTP.
- Each request and response is packaged in a SOAP message—an XML message containing the information that a web service requires to process the message.
- SOAP messages are written in XML so that they're computer readable, human readable and platform independent.

- SOAP supports an extensive set of types—the primitive types, as well as `DateTime`, `XmlNode` and others. SOAP can also transmit arrays of these types.
- When a program invokes a method of a SOAP web service, the request and all relevant information are packaged in a SOAP message enclosed in a SOAP envelope and sent to the server on which the web service resides.
- When a web service receives a SOAP message, it parses the XML representing the message, then processes the message's contents. The message specifies the method that the client wishes to execute and the arguments the client passed to that method.
- After a web service parses a SOAP message, it calls the appropriate method with the specified arguments (if any), and sends the response back to the client in another SOAP message. The client parses the response to retrieve the method's result.

Section 23.4 Representational State Transfer (REST)

- Representational State Transfer (REST) refers to an architectural style for implementing web services. Such web services are often called RESTful web services. Though REST itself is not a standard, RESTful web services are implemented using web standards.
- Each operation in a RESTful web service is identified by a unique URL.
- REST can return data in formats such as XML, JSON, HTML, plain text and media files.

Section 23.5 JavaScript Object Notation (JSON)

- JavaScript Object Notation (JSON) is an alternative to XML for representing data.
- JSON is a text-based data-interchange format used to represent objects in JavaScript as collections of name/value pairs represented as `Strings`.
- JSON is a simple format that makes objects easy to read, create and parse, and allows programs to transmit data efficiently across the Internet because it is much less verbose than XML.
- Each value in a JSON array can be a string, a number, a JSON object, `true`, `false` or `null`.

Section 23.6 Publishing and Consuming SOAP-Based WCF Web Services

- Enabling a web service for client usage is also known as publishing the web service.
- Using a web service is also known as consuming the web service.

Section 23.6.1 Creating a WCF Web Service

- To create a SOAP-based WCF web service in Visual Web Developer, you first create a project of type **WCF Service**. SOAP is the default protocol for WCF web services, so no special configuration is required to create SOAP-based services.
- Visual Web Developer automatically generates files for a **WCF Service** project, including an SVC file, which provides access to the service, and a `Web.config` file, which specifies the service's binding and behavior, and code files for the WCF service class and any other code that is part of the WCF service implementation. In the service class, you define the methods that your WCF web service makes available to client applications.

Section 23.6.2 Code for the `WelcomeSOAPXMLService`

- The service interface describes the service's contract—the set of methods and properties the client uses to access the service.
- The `ServiceContract` attribute exposes a class that implements the service interface as a WCF web service.
- The `OperationContract` attribute exposes a method for remote calls.

Section 23.6.3 Building a SOAP WCF Web Service

- By default, a new code-behind file implements an interface named `IService` that is marked with the `ServiceContract` and `OperationContract` attributes. In addition, the `IService.vb` file defines a class named `CompositeType` with a `DataContract` attribute. The interface contains two sample service methods named `GetData` and `GetDataUsingContract`. The `Service.vb` file contains the code that defines these methods.
- The `Service.svc` file, when accessed through a web browser, provides access to information about the web service.
- When you display the SVC file in the **Solution Explorer**, you see the programming language in which the web service's code-behind file is written, the `Debug` attribute, the name of the service and the code-behind file's location.
- If you change the code-behind file name or the class name that defines the web service, you must modify the SVC file accordingly.

Section 23.6.4 Deploying the `WelcomeSOAPXMLService`

- You can choose **Build Web Site** from the **Build** menu to ensure that the web service compiles without errors. You can also test the web service directly from Visual Web Developer by selecting **Start Without Debugging** from the **Debug** menu. This opens a browser window that contains the SVC page. Once the service is running, you can also access the SVC page from your browser by typing the URL in a web browser.
- By default, the ASP.NET Development Server assigns a random port number to each website it hosts. You can change this behavior by going to the **Solution Explorer** and clicking on the project name to view the **Properties** window. Set the **Use dynamic ports** property to **False** and specify the port number you want to use, which can be any unused TCP port. You can also change the service's virtual path, perhaps to make the path shorter or more readable.
- Web services normally contain a service description that conforms to the Web Service Description Language (WSDL)—an XML vocabulary that defines the methods a web service makes available and how clients interact with them. WSDL documents help applications determine how to interact with the web services described in the documents.
- When viewed in a web browser, an SVC file presents a link to the service's WSDL file and information on using the utility `svcutil.exe` to generate test console applications.
- When a client requests the WSDL URL, the server autogenerates the WSDL that describes the web service and returns the WSDL document.
- Many aspects of web-service creation and consumption—such as generating WSDL files and proxy classes—are handled by Visual Web Developer, Visual Basic 2010 and WCF.

Section 23.6.5 Creating a Client to Consume the `WelcomeSOAPXMLService`

- An application that consumes a SOAP-based web service consists of a proxy class representing the web service and a client application that accesses the web service via a proxy object. The proxy object passes arguments from the client application to the web service as part of the web-service method call. When the method completes its task, the proxy object receives the result and parses it for the client application.
- A proxy object communicates with the web service on the client's behalf. The proxy object is part of the client application, making web-service calls appear to interact with local objects.
- To add a proxy class, right click the project name in the **Solution Explorer** and select **Add Service Reference...** to display the **Add Service Reference** dialog. In the dialog, enter the URL of the service's `.svc` file in the **Address** field. The tools will automatically use that URL to request the web

service's WSDL document. You can rename the service reference's namespace by changing the **Namespace** field. Click the **OK** button to add the service reference.

- A proxy object handles the networking details and the formation of SOAP messages. Whenever the client application calls a web method, the application actually calls a corresponding method in the proxy class. This method has the same name and parameters as the web method that is being called, but formats the call to be sent as a request in a SOAP message. The web service receives this request as a SOAP message, executes the method call and sends back the result as another SOAP message. When the client application receives the SOAP message containing the response, the proxy class deserializes it and returns the results as the return value of the web method that was called.

Section 23.7.2 Creating a REST-Based XML WCF Web Service

- `WebGet` maps a method to a unique URL that can be accessed via an HTTP `GET` operation.
- `WebGet`'s `UriTemplate` property specifies the URI format that is used to invoke a method.
- You can test a REST-based service method using a web browser by going to the `Service.svc` file's network address and appending to the address the URI template with the appropriate arguments.

Section 23.7.3 Consuming a REST-Based XML WCF Web Service

- The `WebClient` class invokes a web service and receives its response.
- `WebClient`'s `DownloadStringAsync` method invokes a web service asynchronously. The `DownloadStringCompleted` event occurs when the `WebClient` receives the completed response from the web service.
- If a service is invoked asynchronously, the application can continue executing and the user can continue interacting with it while waiting for a response from the web service. `DownloadStringCompletedEventArgs` contains the information returned by the web service. We can use this variable's properties to get the returned XML document and any errors that may have occurred during the process.

Section 23.8.1 Creating a REST-Based JSON WCF Web Service

- By default, a web-service method with the `WebGet` attribute returns data in XML format. To return data in JSON format, set `WebGet`'s `ResponseFormat` property to `WebMessageFormat.Json`.
- Objects being converted to JSON must have `Public` properties. This enables the JSON serialization to create name/value pairs that represent each `Public` property and its corresponding value.
- The `DataContract` attribute exposes a class to the client access.
- The `DataMember` attribute exposes a property of this class to the client.
- When we test the web service using a web browser, the response prompts you to download a text file containing the JSON formatted data. You can see the service response as a JSON object by opening the file in a text editor such as Notepad.

Section 23.8.2 Consuming a REST-Based JSON WCF Web Service

- XML serialization converts a custom type into XML data format.
- JSON serialization converts a custom type into JSON data format.
- The `System.Runtime.Serialization.Json` library's `DataContractJsonSerializer` class serializes custom types as JSON objects. To use the `System.Runtime.Serialization.Json` library, you must include a reference to the `System.ServiceModel.Web` assembly in the project.
- Attribute `Serializable` indicates that a class can be used in serialization.

- A `MemoryStream` object is used to encapsulate the JSON object so we can read data from the byte array using stream semantics. The `MemoryStream` object is read by the `DataContractJsonSerializer` and then converted into a custom type.

Section 23.9 Blackjack Web Service: Using Session Tracking in a SOAP-Based WCF Web Service

- Using session tracking eliminates the need for information about the client to be passed between the client and the web service multiple times.

Section 23.9.1 Creating a Blackjack Web Service

- Web services store session information to provide more intuitive functionality.
- A service's interface uses a `ServiceContract` with the `SessionMode` property set to `Required` to indicate that the service needs a session to run. The `SessionMode` property is `Allowed` by default and can also be set to `NotAllowed` to disable sessions.
- Setting the `ServiceBehavior`'s `InstanceContextMode` property to `PerSession` creates a new instance of the class for each session. The `InstanceContextMode` property can also be set to `PerCall` or `Single`. `PerCall` uses a new object of the web-service class to handle every method call to the service. `Single` uses the same object of the web-service class to handle all calls to the service.

Section 23.10 Airline Reservation Web Service: Database Access and Invoking a Service from ASP.NET

- You can add a database and corresponding LINQ to SQL classes to create a `DataContext` object to support database operations of your web service.

Section 23.11 Equation Generator: Returning User-Defined Types

- Instances of user-defined types can be passed to or returned from web-service methods.

Terminology

adding a service reference to a project in Visual Web Developer
address of a WCF service
binding of a WCF service
consuming a web service
contract of a WCF service
`DataContract` attribute
`DataContractJsonSerializer` class
`DataMember` attribute
`DownloadStringCompletedEventArgs` class
endpoint (of a WCF service)
`endpointBehaviors` element in `web.config`
`Error` property of `DownloadStringCompletedEventArgs`
firewall
`get` request (HTTP)
HTTP (Hypertext Transfer Protocol)
request type
`InstanceContextMode` property of `ServiceBehavior` attribute
JavaScript Object Notation (JSON)
JSON serialization
`OperationContract` attribute
`post` request (HTTP)
`protocolMapping` element in `web.config`
proxy class for a web service
publishing a web service
query string
Representational State Transfer (REST)
request method
`ResponseFormat` property of the `WebGet` attribute
RESTful web services
`Result` property of `DownloadStringCompletedEventArgs`
serialization
server response
server-side form handler
service description for a web service
`ServiceBehavior` attribute
`ServiceContract` attribute
`SessionMode` property of `ServiceContract` attribute
Simple Object Access Protocol (SOAP)

SOAP (Simple Object Access Protocol)
SVC file
`UriTemplate` property of `WebGet` attribute
user-defined types in web services
WCF service endpoint
WCF Web Service project type
Web service
Web Service Description Language (WSDL)
web service host
`WebClient`
`webHttp` `Web.config` property
`webHttpBinding` `Web.config` binding setting
WSDL (Web Service Description Language)

Self-Review Exercises

23.1 State whether each of the following is *true* or *false*. If *false*, explain why.

a) The purpose of a web service is to create objects of a class located on a web service host. This class then can be instantiated and used on the local machine.
b) You must explicitly create the proxy class after you add a service reference for a SOAP-based service to a client application.
c) A client application can invoke only those methods of a web service that are tagged with the `OperationContract` attribute.
d) To enable session tracking in a web-service method, no action is required other than setting the `SessionMode` property to `SessionMode.Required` in the `ServiceContract` attribute.
e) Operations in a REST web service are defined by their own unique URLs.
f) A SOAP-based web service can return data in JSON format.
g) For a client application to deserialize a JSON object, the client must define a `Serializable` class with public instance variables or properties that match those serialized by the web service.

23.2 Fill in the blanks for each of the following statements:

a) A key difference between SOAP and REST is that SOAP messages have data wrapped in a(n) _____.
b) A WCF web service exposes its methods to clients by adding the _____ and _____ attributes to the service interface.
c) Web-service requests are typically transported over the Internet via the _____ protocol.
d) To return data in JSON format from a REST-based web service, the _____ property of the `WebGet` attribute is set to _____.
e) _____ transforms an object into a format that can be sent between a web service and a client.
f) To parse a HTTP response in XML data format, the client application must import the response's _____.

Answers to Self-Review Exercises

23.1 a) False. Web services are used to execute methods on web service hosts. The web service receives the arguments it needs to execute a particular method, executes the method and returns the result to the caller. b) False. The proxy class is created by Visual Basic or Visual Web Developer when you add a Service Reference to your project. The proxy class itself is hidden from you. c) True. d) True. e) True. f) False. A SOAP web service implicitly returns data in XML format. g) True.

23.2 a) envelope. b) `ServiceContract`, `OperationContract`. c) HTTP. d) `ResponseFormat`, `WebMessageFormat.Json`. e) Serialization. f) namespace.

Exercises

23.3 *(Phone-Book Web Service)* Create a REST-based web service that stores phone-book entries in a database (`PhoneBook.mdf`, which is provided in the examples directory for this chapter) and a client application that consumes this service. Give the client user the capability to enter a new contact (service method `AddEntry`) and to find contacts by last name (service method `GetEntries`). Pass only primitive types as arguments to the web service. Add a `DataContext` to the web-service project to enable the web service to interact with the database. The `GetEntries` method should return an array of `String`s that contains the matching phone-book entries. Each `String` in the array should consist of the last name, first name and phone number for one phone-book entry separated by commas. Build an ASP.NET client (Fig. 23.33) to interact with this web service. To use an asynchronous web request from an ASP.NET client, you must set the `Async` property to true by adding `Async="true"` to the `.aspx` page directive. Since the `AddEntry` method accepts a request and does not return a response to the client, you can use `WebClient`'s `OpenRead` method to access the service method. You can use the `ToArray` method on the LINQ query to return an array containing LINQ query results.

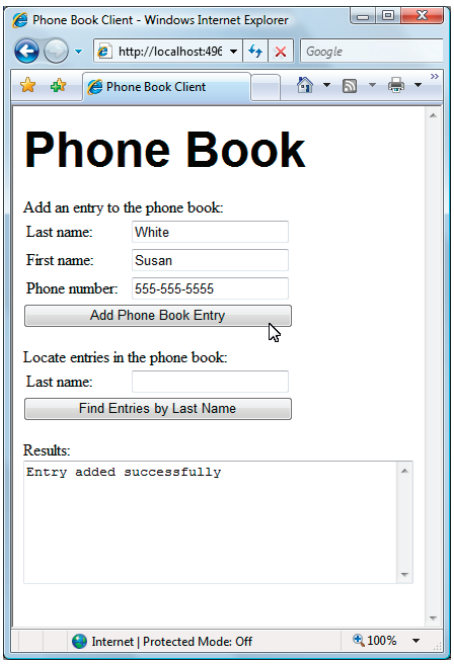


Fig. 23.33 | Template web form for phone book client.

23.4 *(Phone-Book Web Service Modification)* Modify Exercise 23.3 so that it uses a class named `PhoneBookEntry` to represent a row in the database. The web service should return objects of type `PhoneBookEntry` in XML format for the `GetEntries` service method, and the client application should use XML document parsing to interpret the `PhoneBookEntry` object.

23.5 *(Phone-Book Web Service with JSON)* Modify Exercise 23.4 so that the `PhoneBookEntry` class is passed to and from the web service as a JSON object. Use serialization to convert the JSON object into an object of type `PhoneBookEntry`.

23.6 ***(Blackjack Modification)*** Modify the blackjack web-service example in Section 23.9 to include class `Card`. Change service method `DealCard` so that it returns an object of type `Card` and modify method `GetHandValue` to receive an array of `Card`s. Also modify the client application to keep track of what cards have been dealt by using `Card` objects. Your `Card` class should include properties for the face and suit of the card. [*Note:* When you create the `Card` class, be sure to add the `DataContract` attribute to the class and the `DataMember` attribute to the properties. Also, in a SOAP-based service, you don't need to define your own `Card` class on the client as well. The `Card` class will be exposed to the client through the service reference that you add to the client. If the service reference is named `ServiceReference`, you'll access the card type as `ServiceReference.Card`.]

23.7 ***(Airline Reservation Web-Service Modification)*** Modify the airline reservation web service in Section 23.10 so that it contains two separate methods—one that allows users to view all available seats, and another that allows users to reserve a particular seat that is currently available. Use an object of type `Ticket` to pass information to and from the web service. The web service must be able to handle cases in which two users view available seats, one reserves a seat and the second user tries to reserve the same seat, not knowing that it is now taken. The names of the methods that execute should be `Reserve` and `GetAllAvailableSeats`.