Web Services

30

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 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.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

Outline

30.1 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 apps on other computers via common data formats and protocols, such as XML (see Chapter 24), JSON (Section 30.5) 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—the original technology for communicating with web services—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. Though SOAP is still in use today, most web services are developed using REST, as we'll do in this chapter.

You'll build this chapter's web services using Visual Studio Express 2012 for Web. To interact with the web services, you'll build Windows `Forms` client apps using Visual Studio Express 2012 for Windows Desktop. Full versions of Visual Studio 2012 include the functionality of both Express editions. Though we demonstrate only Windows `Forms` client apps, the techniques you'll learn here can be used in any type of app you've implemented in this book.

©2014 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.
©Copyright 1992–2014 by Deitel & Associates, Inc. and Pearson Education, Inc. All Rights Reserved.

30.2 WCF Services Basics

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 host (e.g., `www.deitel.com`) used to access the service.
- A **binding** specifies how a client *communicates* with the service (e.g., SOAP, REST, etc.). 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 computer on which the web service resides is referred to as a **web service host**. The client app 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 app. This kind of *distributed computing* benefits systems in various ways. For example, an app without direct access to data on another system might be able to retrieve this data via a web service. Similarly, an app lacking the processing power necessary to perform specific computations could use a web service to take advantage of another system's superior resources.

30.3 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 sends 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 HTML 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.

Sending Data in a `get` Request

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

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

Sending Data in a `post` *Request*
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.

30.4 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. In browser-based apps, the results of a particular REST 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. REST web services typically return data in XML (Chapter 24) or JSON (Section 30.5) format, but can return other formats, such as HTML, plain text and media files.

30.5 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 `string`s. It's commonly used in Ajax apps. 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.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

30.6 Publishing and Consuming REST-Based XML Web Services

This section presents our first example of **publishing** (enabling for client access) and **consuming** (using) a web service using REST architecture.

30.6.1 WCF Web Service Project

To build your web service, you'll create a **WCF Service Application** project. The IDE will generate various files, including an **SVC file** (`Service1.svc`, which provides access to the service) and a **`Web.config`** file (which specifies the service's binding and behavior). The IDE will also generate code files for the **WCF service class** and any other code that's part of the WCF service's implementation. In the service class, you'll define the methods that your web service makes available to client apps.

30.6.2 Implementing a REST-Based XML WCF Web Service

Figures 30.1 and 30.2 present the code-behind files for the `WelcomeRESTXMLService` WCF web service that you'll build in Section 30.6.3. When creating services, you work almost exclusively in the code-behind files. The service that you'll build in the next few sections 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. You'll use a parameter in the method definition to demonstrate that a client can send data to a web service.

Web Service Interface

Figure 30.1 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 9) exposes a class that implements this interface as a WCF web service. The **`OperationContract`** attribute (line 13) exposes the `Welcome` method to clients for remote calls. Both of these attributes are defined in namespace `System.ServiceModel` (line 4).

```
1  // Fig. 30.1: IWelcomeRESTXMLService.cs
2  // WCF web service interface. A class that implements this interface
3  // returns a welcome message through REST architecture and XML data format
4  using System.ServiceModel;
5  using System.ServiceModel.Web;
6
7  namespace WelcomeRESTXMLService
8  {
9     [ServiceContract]
10    public interface IWelcomeRESTXMLService
11    {
12       // returns a welcome message
13       [OperationContract]
14       [WebGet( UriTemplate = "/welcome/{yourName}" )]
15       string Welcome( string yourName );
16    } // end interface IWelcomeRESTXMLService
17 } // end namespace WelcomeRESTXMLService
```

Fig. 30.1 | WCF web-service interface. A class that implements this interface returns a welcome message through REST architecture and XML data format.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

WebGet Attribute

The `Welcome` method's **`WebGet`** attribute (line 14; from namespace `System.ServiceModel.Web`) maps a method name and its parameters to a unique URL that can be accessed via an HTTP `get` operation programmatically or in a web browser. `WebGet`'s **`UriTemplate`** property (line 14) specifies the URI format that is used to invoke the method. You access the `Welcome` method in a web browser by appending text that matches the `UriTemplate` to the end of the service's location, as in

```
http://localhost:portNumber/WelcomeRESTXMLService.svc/welcome/Paul
```

Web Service Implementation

`WelcomeRESTXMLService` (Fig. 30.2) is the class that implements the `IWelcomeRESTXMLService` interface that was declared as the `ServiceContract`. Lines 9–14 define the method `Welcome`, which returns a `string` welcoming you to WCF web services. As you'll soon see, the `string` object will be serialized to XML—that is, it will be converted to an XML representation—then returned to the client. In the next section, you'll build the web service from scratch.

```
1  // Fig. 30.2: WelcomeRESTXMLService.svc.cs
2  // WCF web service that returns a welcome message using REST architecture
3  // and XML data format.
4  namespace WelcomeRESTXMLService
5  {
6     public class WelcomeRESTXMLService : IWelcomeRESTXMLService
7     {
8        // returns a welcome message
9        public string Welcome( string yourName )
10       {
11          return string.Format(
12             "Welcome to WCF Web Services with REST and XML, {0}!",
13             yourName );
14       } // end method Welcome
15    } // end class WelcomeRESTXMLService
16 } // end namespace WelcomeRESTXMLService
```

Fig. 30.2 | WCF web service that returns a welcome message using REST architecture and XML data format.

30.6.3 Building a REST WCF Web Service

In the following steps, you create a **WCF Service Application** project for the `WelcomeRESTXMLService` and test it using the IIS Express web server that comes with Visual Studio Express 2012 for Web.

Step 1: Creating the Project

To create a project of type **WCF Service Application**, select **File > Project...** to display the **New Project** dialog (Fig. 30.3). Select the **WCF Service Application** template in the **Visual C# > WCF** templates category. By default, Visual Studio Express 2012 for Web places files on the local machine in a directory named `WcfService1`. Rename this folder to `WelcomeRESTXMLService`, specify where you'd like to save the project, then click **OK** to create it.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

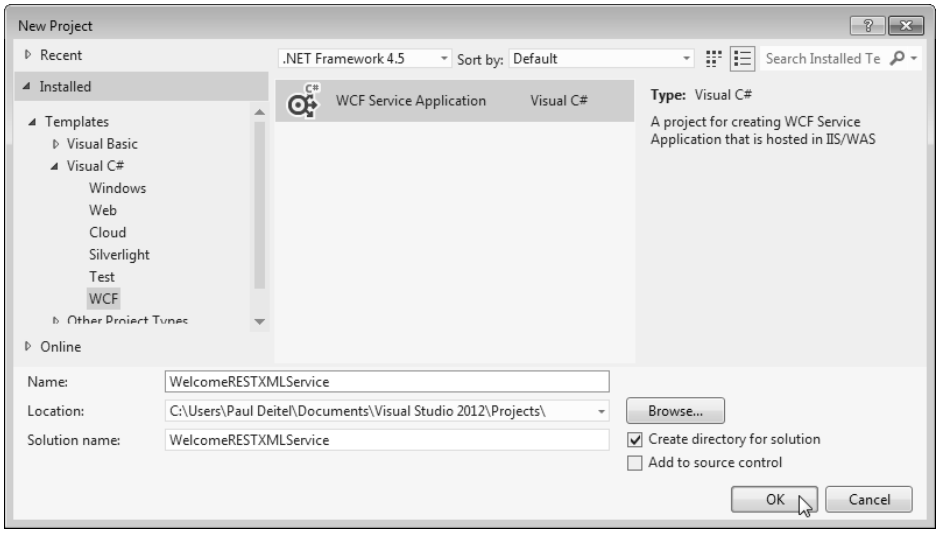


Fig. 30.3 | Creating a **WCF Service Application** in Visual Studio Express 2012 for Web.

Step 2: Examining the Newly Created Project

After creating the project, the code-behind file `Service1.svc.cs`, which contains code for a simple web service, is displayed by default. If it's not displayed, double-click `Service1.svc` in the **Solution Explorer**—the IDE opens the service's code-behind file `Service1.svc.cs`. By default, the code-behind file implements an interface named `IService1`. This interface (located in `IService1.cs`) is marked with the `ServiceContract` and `OperationContract` attributes that we introduced in Section 30.6.2. In addition, the `IService1.cs` file defines a class named `CompositeType` with a `DataContract` attribute (discussed in Section 30.7). The `IService1` interface contains two sample service methods named `GetData` and `GetDataUsingContract`. The `Service1.svc.cs` file defines these methods.

Step 3: Modifying and Renaming the Code-Behind File

To create the `WelcomeRESTXMLService` service developed in this section, first use the IDE's refactoring capabilities to rename the `IService1` interface and the `Service1` class:

1. In `IService1.cs`, right click the interface name `IService1`, then select **Refactor > Rename...**, specify `IWelcomeRESTXMLService` as the new name and click **OK**. In the next dialog, click **Apply** to change the interface's name throughout the project.
2. Repeat the preceding step in `Service1.svc.cs` to rename the class `Service1` as `WelcomeRESTXMLService`.

You should also modify the corresponding file names. In the **Solution Explorer**, rename `IService1.cs` and `Service1.svc` to `IWelcomeRESTXMLService.cs` and `WelcomeRESTXMLService.svc`, respectively. When you rename the SVC file the IDE automatically renames the code-behind file. Next, replace the code in `WelcomeRESTXMLService.svc.cs` with the code in Fig. 30.2.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

Step 4: Modifying the **`Web.config`** *File to Enable REST Architecture*
Figure 30.4 shows part of the default `Web.config` file reformatted and modified to enable the web service to support REST architecture—by default WCF web services use SOAP. All of the changes are located in the `Web.config` file's `system.serviceModel` element. The **`endpointBehaviors`** element (lines 16–20) 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. In the `protocolMapping` element (lines 22–25), line 23 changes the default protocol for communicating with this web service (normally SOAP) to **`webHttpBinding`**, which is used for REST-based HTTP requests.

```
1  <system.serviceModel>
2     <behaviors>
3        <serviceBehaviors>
4           <behavior>
5              <!-- To avoid disclosing metadata information, set the
6                 values below to false before deployment -->
7              <serviceMetadata httpGetEnabled="true"
8                 httpsGetEnabled="true"/>
9              <!-- To receive exception details in faults for debugging
10                purposes, set the value below to true.  Set to false
11                before deployment to avoid disclosing exception
12                information -->
13             <serviceDebug includeExceptionDetailInFaults="false"/>
14          </behavior>
15       </serviceBehaviors>
16       <endpointBehaviors>
17          <behavior>
18             <webHttp/>
19          </behavior>
20       </endpointBehaviors>
21    </behaviors>
22    <protocolMapping>
23       <add binding="webHttpBinding" scheme="http" />
24       <add binding="basicHttpsBinding" scheme="https" />
25    </protocolMapping>
26    <serviceHostingEnvironment aspNetCompatibilityEnabled="true"
27       multipleSiteBindingsEnabled="true"/>
28 </system.serviceModel>
```

Fig. 30.4 | The modified `system.serviceModel` element of the `WelcomeRESTXMLService` project's `Web.config` file.

30.6.4 Deploying the `WelcomeRESTXMLService`

Choose **Build Solution** from the **Build** menu to ensure that the web service compiles without errors. To deploy the web service on IIS Express for testing, simply right click the web service's SVC file, then select **View in Browser**. A browser window opens and displays information about the service (Fig. 30.5), which is your confirmation that the service was deployed and can now receive requests. The information shown in the web browser is generated dynamically when the `WelcomeRESTXMLService.svc` file is requested.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

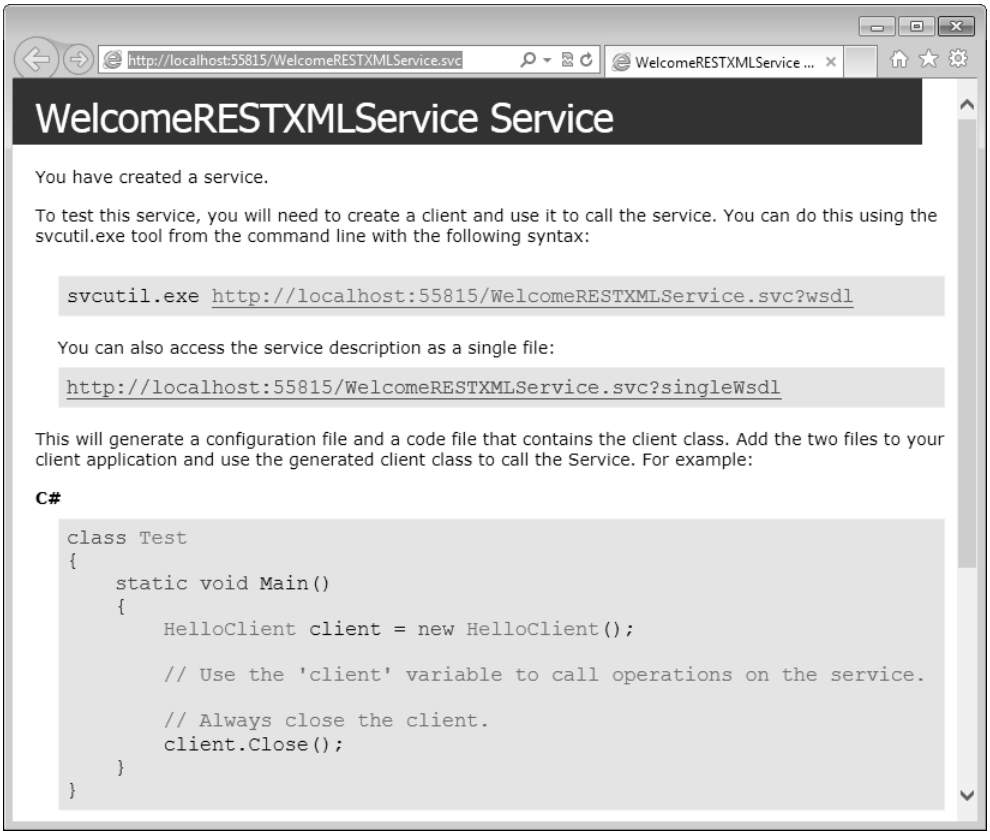


Fig. 30.5 | SVC file rendered in a web browser.

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

```
http://localhost:portNumber/SVCFileName
```

(See the actual URL in Fig. 30.5.) By default, IIS Express assigns a random port number to each website or web service it hosts.

Testing the Web Service

To test the web service's `Welcome` method, add

```
/welcome/YourName
```

to the end of the URL in Fig. 30.5, then press *Enter*. Figure 30.6 shows the result. This change to the URL uses the `UriTemplate` (line 14 of Fig. 30.1) to invoke method `Welcome` with the argument `Paul`. The browser displays the XML response from the web service. In Section 30.6.5, you'll learn how to consume this service from an app.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

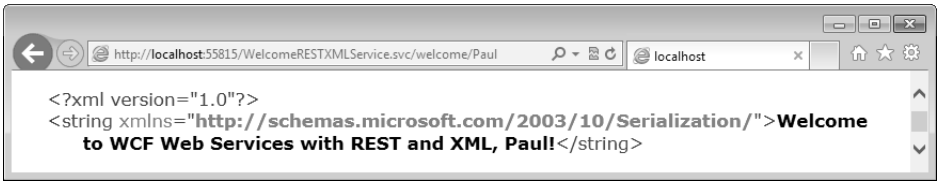


Fig. 30.6 | Response from `WelcomeRESTXMLService` in XML data format.

30.6.5 Consuming a REST-Based XML WCF Web Service

`WelcomeRESTXMLClient` (Fig. 30.7) is a Windows `Forms` app that uses class **`HttpClient`** (line 13) from namespace `System.Net.Http` (line 4) to invoke the web service and receive its response. The `System.Net.Http` namespace is not part of the project by default. To add a reference to its assembly, right click the `References` folder in the **Solution Explorer**, select **Add Reference...** to display the **Reference Manager** dialog, place a check in the checkbox for `System.Net.Http` then click **OK**.

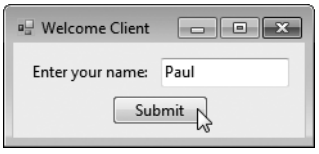
```
1  // Fig. 30.7: WelcomeRESTXML.cs
2  // Client that consumes the WelcomeRESTXMLService.
3  using System;
4  using System.Net.Http;
5  using System.Windows.Forms;
6  using System.Xml.Linq;
7
8  namespace WelcomeRESTXMLClient
9  {
10    public partial class WelcomeRESTXML : Form
11    {
12       // object to invoke the WelcomeRESTXMLService
13       private HttpClient client = new HttpClient();
14
15       private XNamespace xmlNamespace = XNamespace.Get(
16          "http://schemas.microsoft.com/2003/10/Serialization/" );
17
18       public WelcomeRESTXML()
19       {
20          InitializeComponent();
21       } // end constructor
22
23       // get user input, pass it to web service, and process response
24       private async void submitButton_Click( object sender, EventArgs e )
25       {
26          // send request to WelcomeRESTXMLService
27          string result = await client.GetStringAsync( new Uri(
28             "http://localhost:55815/WelcomeRESTXMLService.svc/welcome/" +
29             textBox.Text ) );
30
31          // parse the returned XML
32          XDocument xmlResponse = XDocument.Parse( result );
```

Fig. 30.7 | Client that consumes the `WelcomeRESTXMLService`. (Part 1 of 2.)

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

```
33
34          // get the <string> element's value
35          MessageBox.Show( xmlResponse.Element(
36             xmlNamespace + "string" ).Value, "Welcome" );
37       } // end method submitButton_Click
38    } // end class WelcomeRESTXML
39 } // end namespace WelcomeRESTXMLClient
```

a) User inputs name



b) Message sent from **WelcomeRESTXMLService**

Fig. 30.7 | Client that consumes the `WelcomeRESTXMLService`. (Part 2 of 2.)

Recall that we introduced C#'s `async`/`await` mechanism in Chapter 28. This mechanism is ideal for invoking web services. In this example, we declare the **Submit** button's event handler `async` then `await` a call to `HttpClient`'s method **`GetStringAsync`**, which invokes the web service and returns its response. The method's argument (i.e., 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. The URL's port number must match the one issued to the web service by IIS Express—*this might be different on your computer*, so you might need to modify the port number in line 28. When the call to the web service completes, the event handler continues executing at line 32, which parses the XML response using `XDocument` method `Parse`. In lines 15–16, we specify the XML message's namespace (the value of `xmlns` in Fig. 30.6). This is required to access the elements in the service's XML response so that we can display the welcome `string` in a `MessageBox` (lines 35–36).

30.7 Publishing and Consuming REST-Based JSON Web Services

You'll now reimplement the example from Section 30.6 to build a RESTful web service that returns data in JSON format.

30.7.1 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. 30.8, we modify the `WelcomeRESTXMLService` to return data in *JSON format* by setting `WebGet`'s **`ResponseFormat`** property to `WebMessageFormat.Json` (line 15)—the default is `WebMessageFormat.XML`. For JSON serialization to work properly, the objects being converted to JSON must have *properties* with both `set` and `get` accessors. This enables the JSON serialization to create name/value pairs representing each property and its corresponding value. The previous example returned a `string` object containing the response. Even though `string`s are objects, they do not have any properties that represent

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

their contents. So, lines 21–27 define a `TextMessage` class that encapsulates a `String` value and defines a `Public` property `Message` to access that value. The **`DataContract`** attribute (line 21) exposes the `TextMessage` class to the client. Similarly, the **`DataMember`** attribute (line 25) exposes a property of this class to the client. This property will appear in the JSON object as a name/value pair. Only `DataMember`s of a `DataContract` are serialized.

```
1  // Fig. 30.8: IWelcomeRESTJSONService.cs
2  // WCF web service interface that returns a welcome message through REST
3  // architecture and JSON format.
4  using System.Runtime.Serialization;
5  using System.ServiceModel;
6  using System.ServiceModel.Web;
7
8  namespace WelcomeRESTJSONService
9  {
10    [ServiceContract]
11    public interface IWelcomeRESTJSONService
12    {
13       // returns a welcome message
14       [OperationContract]
15       [WebGet( ResponseFormat = WebMessageFormat.Json,
16          UriTemplate = "/welcome/{yourName}" )]
17       TextMessage Welcome( string yourName );
18    } // end interface IWelcomeRESTJSONService
19
20    // class to encapsulate a string to send in JSON format
21    [DataContract]
22    public class TextMessage
23    {
24       // automatic property message
25       [DataMember]
26       public string Message { get; set; }
27    } // end class TextMessage
28 } // end namespace WelcomeRESTJSONService
```

Fig. 30.8 | WCF web-service interface that returns a welcome message through REST architecture and JSON format.

Figure 30.9 shows the implementation of the interface of Fig. 30.8. The `Welcome` method (lines 9–17) 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 15 in Fig. 30.8) and sent to the client.

```
1  // Fig. 30.9: WelcomeRESTJSONService.cs
2  // WCF web service that returns a welcome message through REST
3  // architecture and JSON format.
4  namespace WelcomeRESTJSONService
5  {
```

Fig. 30.9 | WCF web service that returns a welcome message through REST architecture and JSON format. (Part 1 of 2.)

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

```
6     public class WelcomeRESTJSONService : IWelcomeRESTJSONService
7     {
8        // returns a welcome message
9        public TextMessage Welcome( string yourName )
10       {
11          // add welcome message to field of TextMessage object
12          TextMessage message = new TextMessage();
13          message.Message = string.Format(
14             "Welcome to WCF Web Services with REST and JSON, {0}!",
15             yourName );
16          return message;
17       } // end method Welcome
18    } // end class WelcomeRESTJSONService
19 } // end namespace WelcomeRESTJSONService
```

Fig. 30.9 | WCF web service that returns a welcome message through REST architecture and JSON format. (Part 2 of 2.)

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*`.json`, which is a text file that contains the JSON formatted data. By opening the file in a text editor such as Notepad (Fig. 30.10), you can see the service response as a JSON object. Notice that the property named `Message` has the welcome message as its value.

Fig. 30.10 | Response from `WelcomeRESTJSONService` in JSON data format.

30.7.2 Consuming a REST-Based JSON WCF Web Service

Objects of class types that are sent as arguments to or returned from a REST web service are converted to XML or JSON data format—known as **XML serialization** or **JSON serialization**, respectively. In Fig. 30.11, we consume the `WelcomeRESTJSONService` service using an object of the `System.Runtime.Serialization.Json` namespace's **`DataContractJsonSerializer`** class (lines 31–32). (You must add a reference to the `System.Runtime.Serialization` assembly.) `DataContractJsonSerializer` deserializes the JSON response and assigns its fields to an object of the `TextMessage` class (lines 43–47). The **`Serializable`** attribute (line 43) indicates that `TextMessage` objects can be converted to and from JSON format. Class `TextMessage` 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 32). In lines 33–35, we use the `System.Text` namespace's `Encoding.Unicode.GetBytes` method to convert the JSON response to a Uni-

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

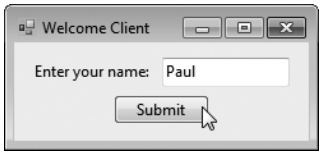
code encoded byte array, then encapsulate the array in a `MemoryStream` object so we can read data from the array using stream semantics. The bytes in the `MemoryStream` object are read by the `DataContractJsonSerializer` and deserialized into a `TextMessage` object.

```
1  // Fig. 30.11: WelcomeRESTJSONForm.cs
2  // Client that consumes the WelcomeRESTJSONService.
3  using System;
4  using System.IO;
5  using System.Net.Http;
6  using System.Runtime.Serialization.Json;
7  using System.Text;
8  using System.Windows.Forms;
9
10 namespace WelcomeRESTJSONClient
11 {
12    public partial class WelcomeRESTJSONForm : Form
13    {
14       // object to invoke the WelcomeRESTJSONService
15       private HttpClient client = new HttpClient();
16
17       public WelcomeRESTJSONForm()
18       {
19          InitializeComponent();
20       } // end constructor
21
22       // get user input and pass it to the web service
23       private void submitButton_Click( object sender, EventArgs e )
24       {
25          // send request to WelcomeRESTJSONService
26          string result = await client.GetStringAsync( new Uri(
27             "http://localhost:56429/WelcomeRESTJSONService.svc/welcome/" +
28             textBox.Text ) );
29
30          // deserialize response into a TextMessage object
31          DataContractJsonSerializer JSONSerializer =
32             new DataContractJsonSerializer( typeof( TextMessage ) );
33          TextMessage message =
34             ( TextMessage ) JSONSerializer.ReadObject( new
35             MemoryStream( Encoding.Unicode.GetBytes( e.Result ) ) );
36
37          // display Message text
38          MessageBox.Show( message.Message, "Welcome" );
39       } // end method submitButton_Click
40    } // end class WelcomeRESTJSONForm
41
42    // TextMessage class representing a JSON object
43    [Serializable]
44    public class TextMessage
45    {
46       public string Message;
47    } // end class TextMessage
48 } // end namespace WelcomeRESTJSONClient
```

Fig. 30.11 | Client that consumes the `WelcomeRESTJSONService`. (Part 1 of 2.)

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

a) User inputs name.



b) Message sent from `WelcomeRESTJSONService`.

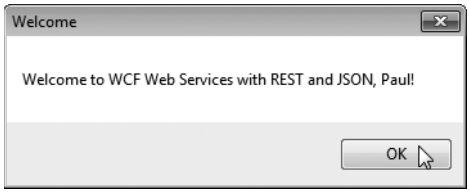


Fig. 30.11 | Client that consumes the `WelcomeRESTJSONService`. (Part 2 of 2.)

30.8 Equation Generator: Returning User-Defined Types

The web services we've demonstrated so far each receive simple `string`s as arguments. It's also possible to process instances of programmer-defined class 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 app 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 0 to 9, 2 specifies equations involving numbers from 10 to 99, and 3 specifies equations containing numbers from 100 to 999). The web service then generates an equation consisting of random numbers in the proper range. The client app receives the `Equation` and displays the sample question to the user.

Defining Class `Equation`

We define class `Equation` in Fig. 30.12. Lines 35–55 define a constructor that takes three arguments—two `int`s representing the left and right operands and a `string` that represents the arithmetic operation to perform. The constructor sets the `Equation`'s properties, then calculates the appropriate result. The parameterless constructor (lines 28–32) calls the three-argument constructor (lines 35–55) and passes default values.

```
1  // Fig. 30.12: Equation.cs
2  // Class Equation that contains information about an equation.
3  using System.Runtime.Serialization;
4
5  namespace EquationGeneratorServiceXML
6  {
7     [DataContract]
8     public class Equation
9     {
10       // automatic property to access the left operand
11       [DataMember]
12       private int Left { get; set; }
13
14       // automatic property to access the right operand
15       [DataMember]
16       private int Right { get; set; }
```

Fig. 30.12 | Class `Equation` that contains information about an equation. (Part 1 of 3.)

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

```
17
18         // automatic property to access the result of applying
19         // an operation to the left and right operands
20         [DataMember]
21         private int Result { get; set; }
22
23         // automatic property to access the operation
24         [DataMember]
25         private string Operation { get; set; }
26
27         // default constructor
28         public Equation()
29            : this( 0, 0, "add" )
30         {
31            // empty body
32         } // end default constructor
33
34         // three-argument constructor for class Equation
35         public Equation( int leftValue, int rightValue, string type )
36         {
37            Left = leftValue;
38            Right = rightValue;
39
40            switch ( type ) // perform appropriate operation
41            {
42               case "add": // addition
43                  Result = Left + Right;
44                  Operation = "+";
45                  break;
46               case "subtract": // subtraction
47                  Result = Left - Right;
48                  Operation = "-";
49                  break;
50               case "multiply": // multiplication
51                  Result = Left * Right;
52                  Operation = "*";
53                  break;
54            } // end switch
55         } // end three-argument constructor
56
57         // return string representation of the Equation object
58         public override string ToString()
59         {
60            return string.Format( "{0} {1} {2} = {4}", Left, Operation,
61               Right, Result );
62         } // end method ToString
63
64         // property that returns a string representing left-hand side
65         [DataMember]
66         private string LeftHandSide
67         {
68            get
69            {
```

Fig. 30.12 | Class `Equation` that contains information about an equation. (Part 2 of 3.)

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

```
70              return string.Format( "{0} {1} {2}", Left, Operation, Right );
71           } // end get
72           set
73           {
74              // empty body
75           } // end set
76        } // end property LeftHandSide
77
78        // property that returns a string representing right-hand side
79        [DataMember]
80        private string RightHandSide
81        {
82           get
83           {
84              return Result.ToString();
85           } // end get
86           set
87           {
88              // empty body
89           } // end set
90        } // end property RightHandSide
91     } // end class Equation
92  } // end namespace EquationGeneratorService
```

Fig. 30.12 | Class `Equation` that contains information about an equation. (Part 3 of 3.)

Class `Equation` defines properties `LeftHandSide` (lines 66–76), `RightHandSide` (lines 80–90), `Left` (line 12), `Right` (line 16), `Result` (line 21) and `Operation` (line 25). The web service client does not need to modify the values of properties `LeftHandSide` and `RightHandSide`. However, a property can be serialized only if it has both a `get` and a `set` accessor—even if the `set` accessor has an empty body. Each property is preceded by the `DataMember` attribute to indicate that it should be serialized. `LeftHandSide` (lines 66–76) returns a `string` representing everything to the left of the equals (=) sign in the equation, and `RightHandSide` (lines 80–90) returns a `string` representing everything to the right of the equals (=) sign. `Left` (line 12) returns the `int` to the left of the operator (known as the left operand), and `Right` (lines 16) returns the `int` to the right of the operator (known as the right operand). `Result` (line 21) returns the solution to the equation, and `Operation` (line 25) 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 58–62) returns a `string` representation of the equation.

30.8.1 Creating the REST-Based XML EquationGenerator Web Service

Figures 30.13 and 30.14 present the interface and class for the `EquationGeneratorService` web service, which creates random, customized `Equations`. This web service contains only method `GenerateEquation` (lines 11–28 of Fig. 30.14), which takes two parameters—a `string` representing the mathematical operation (`"add"`, `"subtract"` or `"multiply"`) and a `string` representing the difficulty level. When line 27 of Fig. 30.14 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 30.8.3. Recall from Section 30.6.3 that you must modify the `Web.config` file to enable REST support as well.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

```
// Fig. 30.13: IEquationGeneratorService.cs
// WCF REST service interface to create random equations based on a
// specified operation and difficulty level.
using System.ServiceModel;
using System.ServiceModel.Web;

namespace EquationGeneratorServiceXML
{
   [ServiceContract]
   public interface IEquationGeneratorService
   {
      // method to generate a math equation
      [OperationContract]
      [WebGet( UriTemplate = "equation/{operation}/{level}" )]
      Equation GenerateEquation( string operation, string level );
   } // end interface IEquationGeneratorService
} // end namespace EquationGeneratorService
```

Fig. 30.13 | WCF REST service interface to create random equations based on a specified operation and difficulty level.

```
// Fig. 30.14: EquationGeneratorService.cs
// WCF REST service to create random equations based on a
// specified operation and difficulty level.
using System;

namespace EquationGeneratorServiceXML
{
   public class EquationGeneratorService : IEquationGeneratorService
   {
      // method to generate a math equation
      public Equation GenerateEquation( string operation, string level )
      {
         // calculate maximum and minimum number to be used
         int maximum =
            Convert.ToInt32( Math.Pow( 10, Convert.ToInt32( level ) ) );
         int minimum = Convert.ToInt32(
            Math.Pow( 10, Convert.ToInt32( level ) - 1 ) );

         Random randomObject = new Random(); // generate random numbers

         // create Equation consisting of two random
         // numbers in the range minimum to maximum
         Equation newEquation = new Equation(
            randomObject.Next( minimum, maximum ),
            randomObject.Next( minimum, maximum ), operation );

         return newEquation;
      } // end method GenerateEquation
   } // end class EquationGeneratorService
} // end namespace EquationGeneratorServiceXML
```

Fig. 30.14 | WCF REST service to create random equations based on a specified operation and difficulty level.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

30.8.2 Consuming the REST-Based XML EquationGenerator Web Service

The `MathTutor` app (Fig. 30.15) 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.

```
1  // Fig. 30.15: MathTutor.cs
2  // Math tutor using EquationGeneratorServiceXML to create equations.
3  using System;
4  using System.Net.Http;
5  using System.Windows.Forms;
6  using System.Xml.Linq;
7
8  namespace MathTutorXML
9  {
10    public partial class MathTutor : Form
11    {
12       private string operation = "add"; // the default operation
13       private int level = 1; // the default difficulty level
14       private string leftHandSide; // the left side of the equation
15       private int result; // the answer
16       private XNamespace xmlNamespace = XNamespace.Get(
17          "http://schemas.datacontract.org/2004/07/" +
18          "EquationGeneratorServiceXML" );
19
20       // object used to invoke service
21       private HttpClient service = new HttpClient();
22
23       public MathTutor()
24       {
25          InitializeComponent();
26       } // end constructor
27
28       // generates new equation when user clicks button
29       private async void generateButton_Click(
30          object sender, EventArgs e )
31       {
32          // send request to EquationGeneratorServiceXML
33          string xmlString = await service.DownloadStringAsync( new Uri(
34             "http://localhost:56770/EquationGeneratorService.svc/" +
35             "equation/" + operation + "/" + level ) );
36
37          // parse response and get LeftHandSide and Result values
38          XDocument xmlResponse = XDocument.Parse( e.Result );
39          leftHandSide = xmlResponse.Element(
40             xmlNamespace + "Equation" ).Element(
41             xmlNamespace + "LeftHandSide" ).Value;
42          result = Convert.ToInt32( xmlResponse.Element(
43             xmlNamespace + "Equation" ).Element(
44             xmlNamespace + "Result" ).Value );
45
```

Fig. 30.15 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 1 of 4.)

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

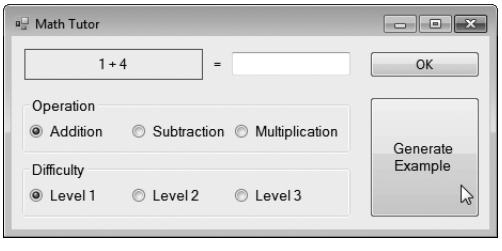
```
46            // display left side of equation
47            questionLabel.Text = leftHandSide;
48            okButton.Enabled = true; // enable okButton
49            answerTextBox.Enabled = true; // enable answerTextBox
50         } // end method generateButton_Click
51
52         // check user's answer
53         private void okButton_Click( object sender, EventArgs e )
54         {
55            if ( !string.IsNullOrEmpty( answerTextBox.Text ) )
56            {
57               // get user's answer
58               int userAnswer = Convert.ToInt32( answerTextBox.Text );
59
60               // determine whether user's answer is correct
61               if ( result == userAnswer )
62               {
63                  questionLabel.Text = string.Empty; // clear question
64                  answerTextBox.Clear(); // clear answer
65                  okButton.Enabled = false; // disable OK button
66                  MessageBox.Show( "Correct! Good job!", "Result" );
67               } // end if
68               else
69               {
70                  MessageBox.Show( "Incorrect. Try again.", "Result" );
71               } // end else
72            } // end if
73         } // end method okButton_Click
74
75         // set the operation to addition
76         private void additionRadioButton_CheckedChanged( object sender,
77            EventArgs e )
78         {
79            if ( additionRadioButton.Checked )
80               operation = "add";
81         } // end method additionRadioButton_CheckedChanged
82
83         // set the operation to subtraction
84         private void subtractionRadioButton_CheckedChanged( object sender,
85            EventArgs e )
86         {
87            if ( subtractionRadioButton.Checked )
88               operation = "subtract";
89         } // end method subtractionRadioButton_CheckedChanged
90
91         // set the operation to multiplication
92         private void multiplicationRadioButton_CheckedChanged(
93            object sender, EventArgs e )
94         {
95            if ( multiplicationRadioButton.Checked )
96               operation = "multiply";
97         } // end method multiplicationRadioButton_CheckedChanged
98
```

Fig. 30.15 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 2 of 4.)

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

```
99          // set difficulty level to 1
100         private void levelOneRadioButton_CheckedChanged( object sender,
101            EventArgs e )
102         {
103            if ( levelOneRadioButton.Checked )
104               level = 1;
105         } // end method levelOneRadioButton_CheckedChanged
106
107         // set difficulty level to 2
108         private void levelTwoRadioButton_CheckedChanged( object sender,
109            EventArgs e )
110         {
111            if ( levelTwoRadioButton.Checked )
112               level = 2;
113         } // end method levelTwoRadioButton_CheckedChanged
114
115         // set difficulty level to 3
116         private void levelThreeRadioButton_CheckedChanged( object sender,
117            EventArgs e )
118         {
119            if ( levelThreeRadioButton.Checked )
120               level = 3;
121         } // end method levelThreeRadioButton_CheckedChanged
122      } // end class MathTutor
123   } // end namespace MathTutorXML
```

a) Generating a level 1 addition equation



b) Answering the question incorrectly

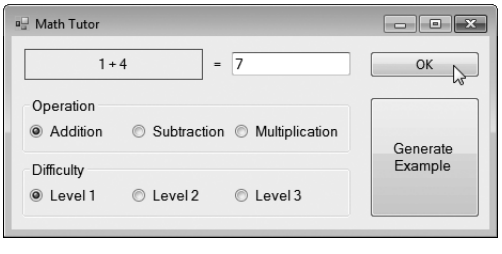


Fig. 30.15 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 3 of 4.)

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

c) Answering the question correctly

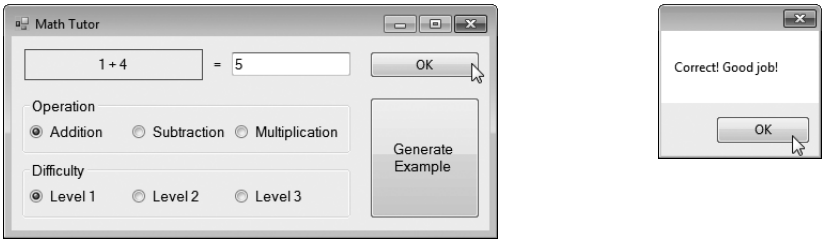


Fig. 30.15 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 4 of 4.)

Difficulty Level and the Arithmetic Operation to Perform

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**. This invokes the corresponding `RadioButton`'s `CheckedChanged` event handler (lines 100–121), which sets integer `level` to the level selected by the user. The default question type is **Addition**, which the user can change by selecting one of the `RadioButtons` in the `GroupBox` labeled **Operation**. This invokes the corresponding operation's event handler in lines 76–97, which assigns to `string operation` the string corresponding to the user's selection.

Invoking the `EquationGeneratorService`

Line 21 defines the `HttpClient` that is used to invoke the web service. Event handler `generateButton_Click` (lines 29–50), which is declared `async`, invokes `EquationGeneratorService` method `GenerateEquation` (line 33–35) using the web service's `UriTemplate` specified at line 14 in Fig. 30.13.

Processing the XML Response

When the response arrives, the await keyword unwraps the XML `string`. Next, line 38 parses the XML response, then lines 39–41 use `XDocument`'s `Element` method to obtain the equation's left side, and lines 42–44 perform the same task to get the equation's result. As in the `WelcomeRESTXMLClient` example, we define the XML response's namespace (lines 16–18) as an `XNamespace` so that we can use it when extracting data from the XML. Then, the handler displays the left-hand side of the equation in `questionLabel` (line 47) and enables `okButton` and `answerTextBox` so that the user can enter an answer.

Checking Whether the User Entered the Correct Answer

When the user clicks **OK**, `okButton_Click` (lines 53–73) checks whether the user provided the correct answer.

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

You can set the web service to return JSON data instead of XML. Figure 30.16 is a modified `IEquationGeneratorService` interface for a service that returns an `Equation` in JSON format. The `ResponseFormat` property (line 14) 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. 30.14. This shows how flexible WCF can be.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

```
1  // Fig. 30.16: IEquationGeneratorService.cs
2  // WCF REST service interface to create random equations based on a
3  // specified operation and difficulty level.
4  using System.ServiceModel;
5  using System.ServiceModel.Web;
6
7  namespace EquationGeneratorServiceJSON
8  {
9     [ServiceContract]
10    public interface IEquationGeneratorService
11    {
12       // method to generate a math equation
13       [OperationContract]
14       [WebGet( ResponseFormat = WebMessageFormat.Json,
15          UriTemplate = "equation/{operation}/{level}" )]
16       Equation GenerateEquation( string operation, string level );
17    } // end interface IEquationGeneratorService
18 } // end namespace EquationGeneratorServiceJSON
```

Fig. 30.16 | WCF REST service interface to create random equations based on a specified operation and difficulty level.

30.8.4 Consuming the REST-Based JSON WCF `EquationGenerator` Web Service

A modified `MathTutor` app (Fig. 30.17) accesses the URI of the `EquationGenerator` web service to get the JSON object (lines 30–32). We define class `Equation` (Fig. 30.18) to use with JSON serialization, so that the `Equation` object sent from the web service can be deserialized into an `Equation` object on the client. The JSON object is deserialized using the `System.Runtime.Serialization.Json` namespace's `DataContractJsonSerializer` (created at lines 35–36) and converted into an `Equation` object (line 37–39). We use the `LeftHandSide` field of the deserialized object (line 42) to display the left side of the equation and the `Result` field (line 53) to obtain the answer.

```
1  // Fig. 30.17: MathTutor.cs
2  // Math tutor using EquationGeneratorServiceJSON to create equations.
3  using System;
4  using System.IO;
5  using System.Net;
6  using System.Runtime.Serialization.Json;
7  using System.Text;
8  using System.Windows.Forms;
9
10 namespace MathTutorJSON
11 {
12    public partial class MathTutor : Form
13    {
14       private string operation = "add"; // the default operation
15       private int level = 1; // the default difficulty level
16       private Equation currentEquation;  // represents the Equation
```

Fig. 30.17 | Math tutor using `EquationGeneratorServiceJSON` to create equations. (Part 1 of 4.)

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

```
17
18          // object used to invoke service
19          private HttpClient service = new HttpClient();
20
21          public MathTutor()
22          {
23             InitializeComponent();
24          } // end constructor
25
26          // generates new equation when user clicks button
27          private async void generateButton_Click(object sender, EventArgs e)
28          {
29             // send request to EquationGeneratorService
30             string jsonString = await service.GetStringAsync( new Uri(
31                "http://localhost:57184/EquationGeneratorService.svc/" +
32                "equation/" + operation + "/" + level ) );
33
34             // deserialize response into an Equation object
35             DataContractJsonSerializer JSONSerializer =
36                new DataContractJsonSerializer( typeof( Equation ) );
37             currentEquation =
38                ( Equation ) JSONSerializer.ReadObject(new MemoryStream(
39                Encoding.Unicode.GetBytes( jsonString ) ) );
40
41             // display left side of equation
42             questionLabel.Text = currentEquation.LeftHandSide;
43             okButton.Enabled = true; // enable okButton
44             answerTextBox.Enabled = true; // enable answerTextBox
45          } // end method generateButton_Click
46
47          // check user's answer
48          private void okButton_Click( object sender, EventArgs e )
49          {
50             if ( !string.IsNullOrEmpty( answerTextBox.Text ) )
51             {
52                // determine whether user's answer is correct
53                if ( currentEquation.Result ==
54                   Convert.ToInt32( answerTextBox.Text ) )
55                {
56                   questionLabel.Text = string.Empty; // clear question
57                   answerTextBox.Clear(); // clear answer
58                   okButton.Enabled = false; // disable OK button
59                   MessageBox.Show( "Correct! Good job!", "Result" );
60                } // end if
61                else
62                {
63                   MessageBox.Show( "Incorrect. Try again.", "Result" );
64                } // end else
65             } // end if
66          } // end method okButton_Click
67
```

Fig. 30.17 | Math tutor using `EquationGeneratorServiceJSON` to create equations. (Part 2 of 4.)

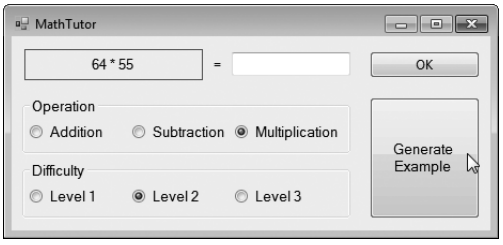
© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

```
68         // set the operation to addition
69         private void additionRadioButton_CheckedChanged( object sender,
70            EventArgs e )
71         {
72            if ( additionRadioButton.Checked )
73               operation = "add";
74         } // end method additionRadioButton_CheckedChanged
75
76         // set the operation to subtraction
77         private void subtractionRadioButton_CheckedChanged( object sender,
78            EventArgs e )
79         {
80            if ( subtractionRadioButton.Checked )
81               operation = "subtract";
82         } // end method subtractionRadioButton_CheckedChanged
83
84         // set the operation to multiplication
85         private void multiplicationRadioButton_CheckedChanged(
86            object sender, EventArgs e )
87         {
88            if ( multiplicationRadioButton.Checked )
89               operation = "multiply";
90         } // end method multiplicationRadioButton_CheckedChanged
91
92         // set difficulty level to 1
93         private void levelOneRadioButton_CheckedChanged( object sender,
94            EventArgs e )
95         {
96            if ( levelOneRadioButton.Checked )
97               level = 1;
98         } // end method levelOneRadioButton_CheckedChanged
99
100        // set difficulty level to 2
101        private void levelTwoRadioButton_CheckedChanged( object sender,
102           EventArgs e )
103        {
104           if ( levelTwoRadioButton.Checked )
105              level = 2;
106        } // end method levelTwoRadioButton_CheckedChanged
107
108        // set difficulty level to 3
109        private void levelThreeRadioButton_CheckedChanged( object sender,
110           EventArgs e )
111        {
112           if ( levelThreeRadioButton.Checked )
113              level = 3;
114        } // end method levelThreeRadioButton_CheckedChanged
115     } // end class MathTutorForm
116  } // end namespace MathTutorJSON
```

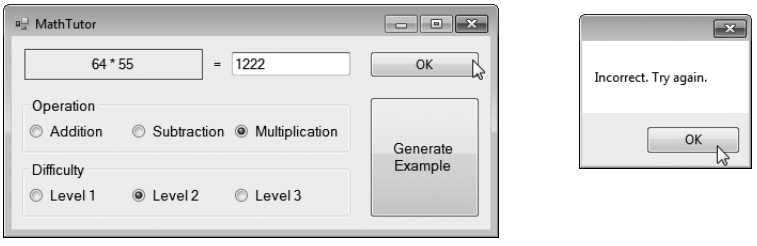
Fig. 30.17 | Math tutor using `EquationGeneratorServiceJSON` to create equations. (Part 3 of 4.)

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

a) Generating a level 2 multiplication equation



b) Answering the question incorrectly



c) Answering the question correctly

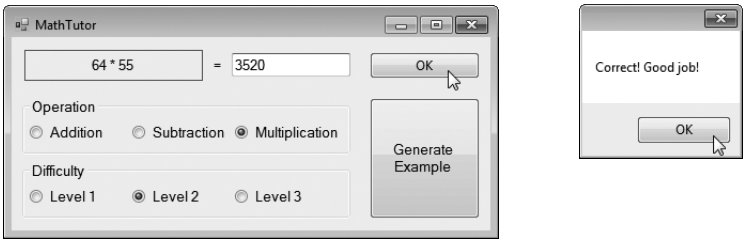


Fig. 30.17 | Math tutor using `EquationGeneratorServiceJSON` to create equations. (Part 4 of 4.)

```
1  // Fig. 30.18: Equation.cs
2  // Equation class representing a JSON object.
3  using System;
4
5  namespace MathTutorJSON
6  {
7     [Serializable]
8     class Equation
9     {
10       public int Left = 0;
11       public string LeftHandSide = null;
12       public string Operation = null;
13       public int Result = 0;
14       public int Right = 0;
15       public string RightHandSide = null;
16    } // end class Equation
17 } // end namespace MathTutorJSON
```

Fig. 30.18 | `Equation` class representing a JSON object.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

30.9 Wrap-Up

This chapter introduced WCF—a set of technologies for building distributed systems in which system components communicate with one another over networks. You used WCF to build web services. 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 WCF facilitates publishing and consuming web services. You learned how to define web services and methods using REST architecture, and how to return data in both XML and JSON formats. You consumed REST-based web services using class `HttpClient` in Windows `Forms` apps.

Self-Review Exercises

30.1 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) A client app can invoke only those methods of a web service that are tagged with the `OperationContract` attribute.

c) Operations in a REST web service are defined by their own unique URLs.

d) A client app can deserialize any JSON object.

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

a) A WCF web service exposes its methods to clients by adding the _____ and _____ attributes to the service interface.

b) Web-service requests are typically transported over the Internet via the _____ protocol.

c) To return data in JSON format from a REST-based web service, the _____ property of the `WebGet` attribute is set to _____.

d) _____ transforms an object into a format that can be sent between a web service and a client.

e) To parse a HTTP response in XML data format, the client app must import the response's _____.

Answers to Self-Review Exercises

30.1 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) True. c) True. d) False. For a client app 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.

30.2 a) `ServiceContract`, `OperationContract`. b) HTTP. c) `ResponseFormat`, `WebMessageFormat.Json`. d) Serialization. e) XML namespace.

Exercises

30.3 ***(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

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

client app 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 simple types as arguments to the web service. Add a `DbContext` 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. 30.19) 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.

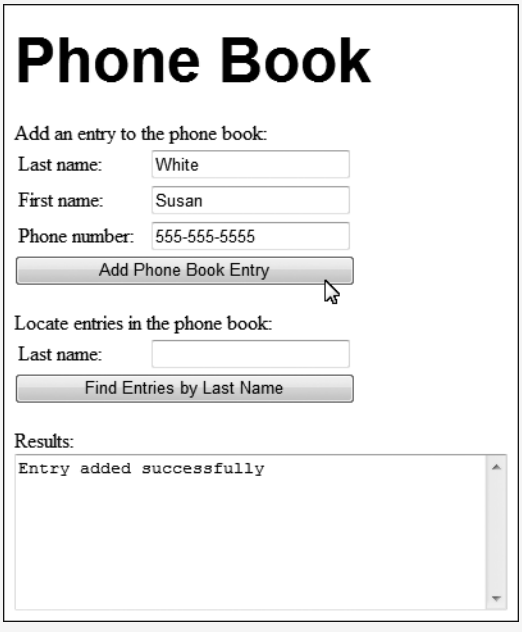


Fig. 30.19 | Template web form for phone book client.

30.4 ***(Phone-Book Web Service Modification)*** Modify Exercise 30.3 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 app should use XML document parsing to interpret the `PhoneBookEntry` object.

30.5 ***(Phone-Book Web Service with JSON)*** Modify Exercise 30.4 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`.

© 2014 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved
©Copyright 1992–2014 by Deitel & Associates, Inc. and Pearson Education, Inc. All Rights Reserved.