Lecture 15a

Windows Communication Foundation (WCF) Web Services

OBJECTIVES

In this lecture you will learn:

- What a WCF service is.
- How to create WCF web services.
- How XML, JSON, XML-Based Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) Architecture enable WCF web services.
- The elements that comprise WCF web services, such as service references, service endpoints, service contracts and service bindings.

OBJECTIVES

- How to create a client that consumes a WCF web service.
- How to use WCF web services with Windows applications 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

1 Introduction

2 WCF Services Basics

3 Simple Object Access Protocol (SOAP)

4 Representational State Transfer (REST)

5 JavaScript Object Notation (JSON)

6 Publishing and Consuming SOAP-Based Web Services

7 Publishing and Consuming REST-Based XML Web Services

8 Publishing and Consuming REST-Based JSON Web Services

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

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

11 Equation Generator: Returning User-Defined Types

1 Introduction

Windows Communication Foundation (WCF) services are a set of technologies for communicating over networks.

WCF uses a common framework for all communication, so you need to learn only one programming model.

A **web service** is **a class** that allows its **methods to be called by methods on other machines** via common data formats and protocols.

1 Introduction (Cont.)

In .NET, method calls are commonly implemented through **Simple Object Access Protocol (SOAP)** or **Representational State Transfer (REST)**.

- SOAP is an **XML-based protocol** of requests and responses.
- REST uses the web's traditional request/response mechanisms such as **GET and POST requests**.

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.

1 Introduction (Cont.)

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**

2 WCF Services Basics

Microsoft's **Windows Communication Foundation** (**WCF**) encompasses several existing technologies.

Each **WCF** **service** has three **key components**:

- 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**.

2 WCF Services Basics (Cont.)

The machine on which the web service resides is the **web service host**.

The **client application sends a method call over a network** to the web service **host**, which processes the call and **returns a response**.

Distributed computing advantages. 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

3 Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is a **platform-independent protocol**.

SOAP messages are contain information in XML. 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-based services **send and receive messages over HTTP connections**.

3 Simple Object Access Protocol (SOAP) (Cont.)

The wire format used to **transmit requests and responses** must support all types passed between the applications.

SOAP types include the **primitive types** (e.g., `Integer`), as well as **`DateTime`**, **`XmlNode`** and others.

SOAP **can also transmit** **arrays** of these types.

3 Simple Object Access Protocol (SOAP) (Cont.)

When a program invokes a method of a SOAP web service, the **request is packaged in a SOAP message**, enclosed in a **SOAP envelope** and sent to the server.

The web service **parses the XML**, 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

The web service **calls the method with the specified arguments** (if any) and web service **sends the response back** to the client in **another SOAP message**.

The client **parses the response** to retrieve the method's result.

4.1 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.

4.1 HTTP get and post Requests

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=FMI
```

search is the name of Google’s server-side form handler, **q** is the **name** of a *variable* in Google’s search form and **FMI** is the **value** 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 (**&**).

4.1 HTTP get and post Requests

Sending Data in a get Request

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.

A **get** request typically limits the query string to a specific number of characters. For example, Internet Explorer restricts the entire URL to **no more than 2083 characters**

4.1 HTTP get and post Requests

Sending Data in a post Request

A **post** request sends form data as part of the **HTTP** message, not as part of the **URL**. 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.

4.2 Representational State Transfer (REST)

Representational State Transfer (**REST**) is an architectural style for implementing web services.

RESTful web services are implemented using web standards. **Each operation** in a **RESTful** web service is **identified by a unique URL**.

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**.

4.2 Representational State Transfer (REST)

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.**

REST web services typically **return data** in **XML** or **JSON** format, but can **return** other formats, such as **HTML**, **plain text** and **media files**.

5 JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) is an **alternative to XML**.

JSON represents objects as **collections of name/value pairs** represented as `String`s.

JSON is a simple format that makes **objects easy to read, create and parse**:

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

5 JavaScript Object Notation (JSON) (Cont.)

Arrays are represented in JSON with square brackets:

[*value1* , *value2* , *value3*]

1. To appreciate the simplicity of JSON, examine this array of address-book entries:

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

6 SOAP-Based Web Services

Following are **5 simple steps** to **develop a SOAP web service**.

Create a **WCF Service Application** Project.

Define the **Service Contract** and **Data Contracts** for **user- defined types of parameters** and **return results of web service methods**

Implement the Web Service

Publish and **Test** the web service

Study (https://www.youtube.com/watch?v=9XvJ_ttnnPA

https://msdn.microsoft.com/en-us/library/ms751519(v=vs.110).aspx

https://msdn.microsoft.com/en-us/library/bb412178(v=vs.110).aspx)

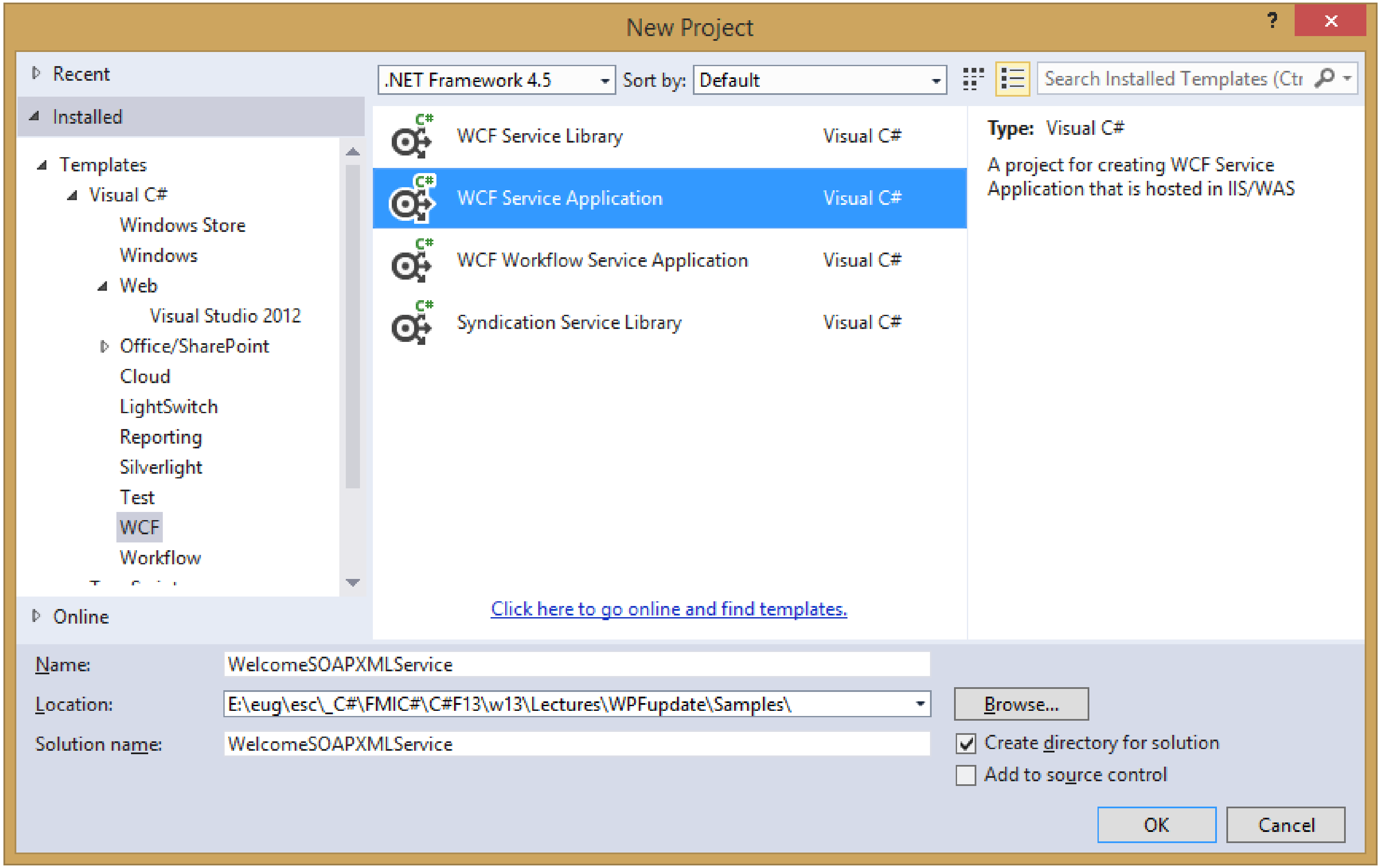
6.1 Publishing SOAP-Based Web Services

6.1 Creating a WCF Web Service

Step 1 Creating the project

To build a SOAP-based web service in Visual Web Developer, create a **WCF Service Application** project.

6.1 Publishing SOAP-Based Web Services



WCF Service Library	Visual C#
WCF Service Application	Visual C#
WCF Workflow Service Application	Visual C#
Syndication Service Library	Visual C#

Type: Visual C#

A project for creating WCF Service Application that is hosted in IIS/WAS

Click here to go online and find templates.

Name: WelcomeSOAPXMLService

Location: E:\eug\esc_C#\FMIC#\C#F13\w13\Lectures\WPFupdate\Samples\

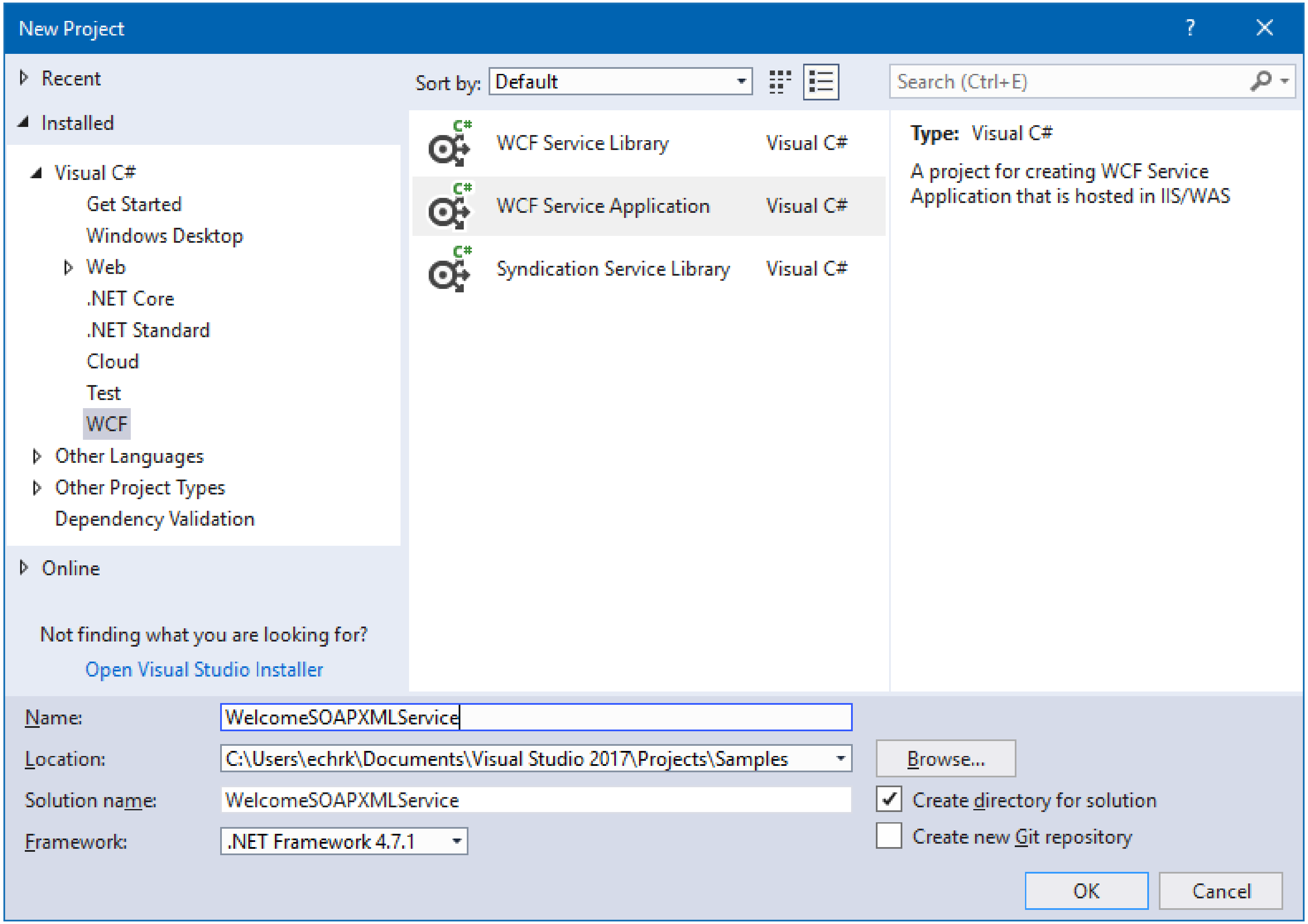
Solution name: WelcomeSOAPXMLService

☑ Create directory for solution

□ Add to source control

Creating a **WCF Service** in Visual Web Developer.

6.1 Publishing SOAP-Based Web Services



Type: Visual C#

A project for creating WCF Service Application that is hosted in IIS/WAS

Name: WelcomeSOAPXMLService

Location: C:\Users\echrk\Documents\Visual Studio 2017\Projects\Samples

Solution name: WelcomeSOAPXMLService

Framework: .NET Framework 4.7.1

☑ Create directory for solution

☐ Create new Git repository

Creating a **WCF Service** in Visual Web Developer.

6.1 Publishing SOAP-Based Web Services

6.1 Creating a WCF Web Service

SOAP is the **default protocol** for WCF web services. No special configuration is required for this protocol.

Visual Web Developer generates files for the WCF

- A **default implementation** of the web **service code**,
- a **SVC file** (`Service.svc`), and
- a `Web.config` file.

6.1 Publishing SOAP-Based Web Services

Step 2 Define the Service Contract

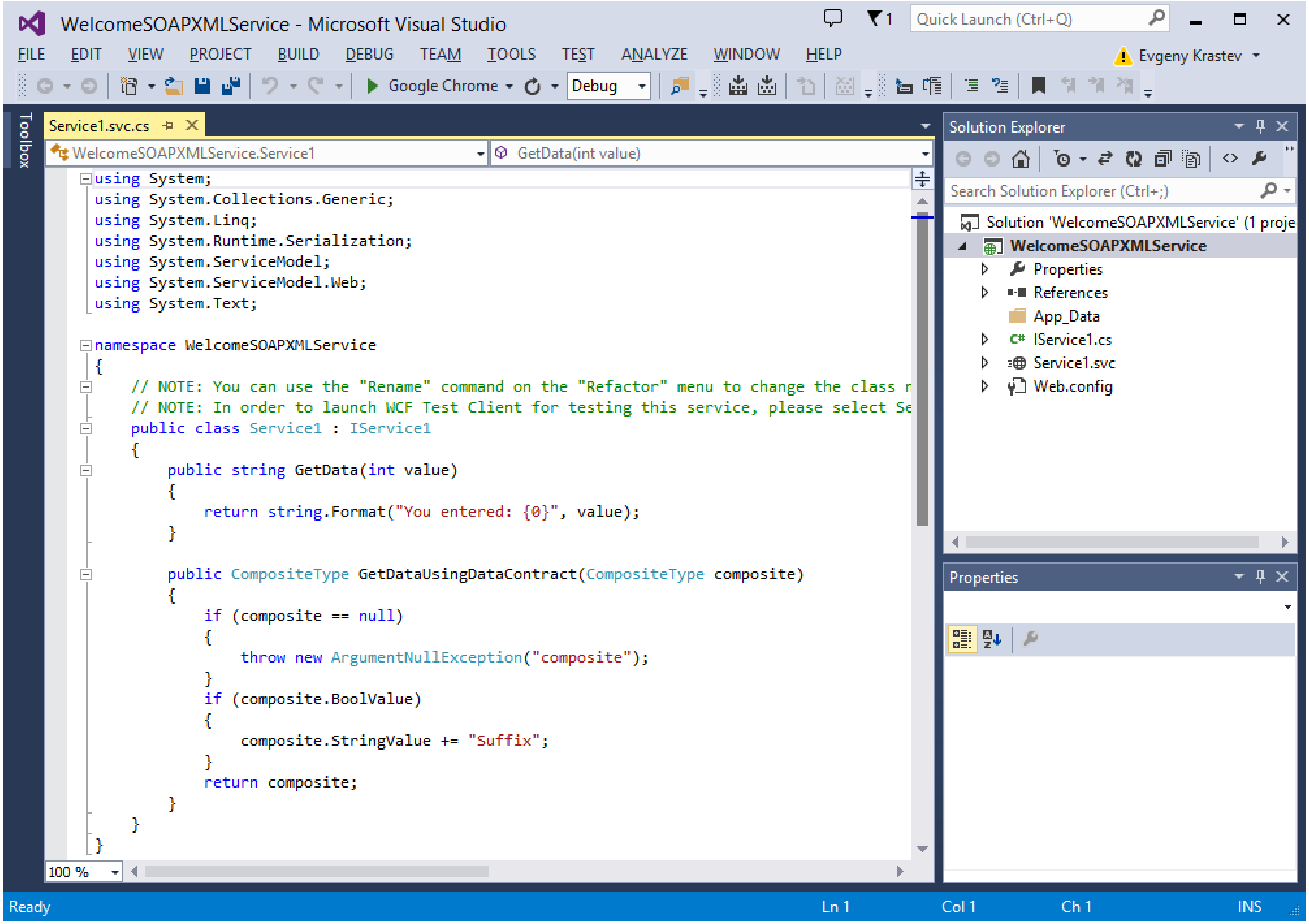
Examine the newly created project.

The **code-behind file** Service1.svc.cs is displayed by **default**. **Contains the code** for the **web service** in `class` Service1.

In the **service class**, you **define the methods** that your WCF web service makes **available to client applications**.

.

6.1 Publishing SOAP-Based Web Services



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace WelcomeSOAPXMLService
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to change the class r
    // NOTE: In order to launch WCF Test Client for testing this service, please select Se
    public class Service1 : IService1
    {
        public string GetData(int value)
        {
            return string.Format("You entered: {0}", value);
        }

        public CompositeType GetDataUsingDataContract(CompositeType composite)
        {
            if (composite == null)
            {
                throw new ArgumentNullException("composite");
            }
            if (composite.BoolValue)
            {
                composite.StringValue += "Suffix";
            }
            return composite;
        }
    }
}
```

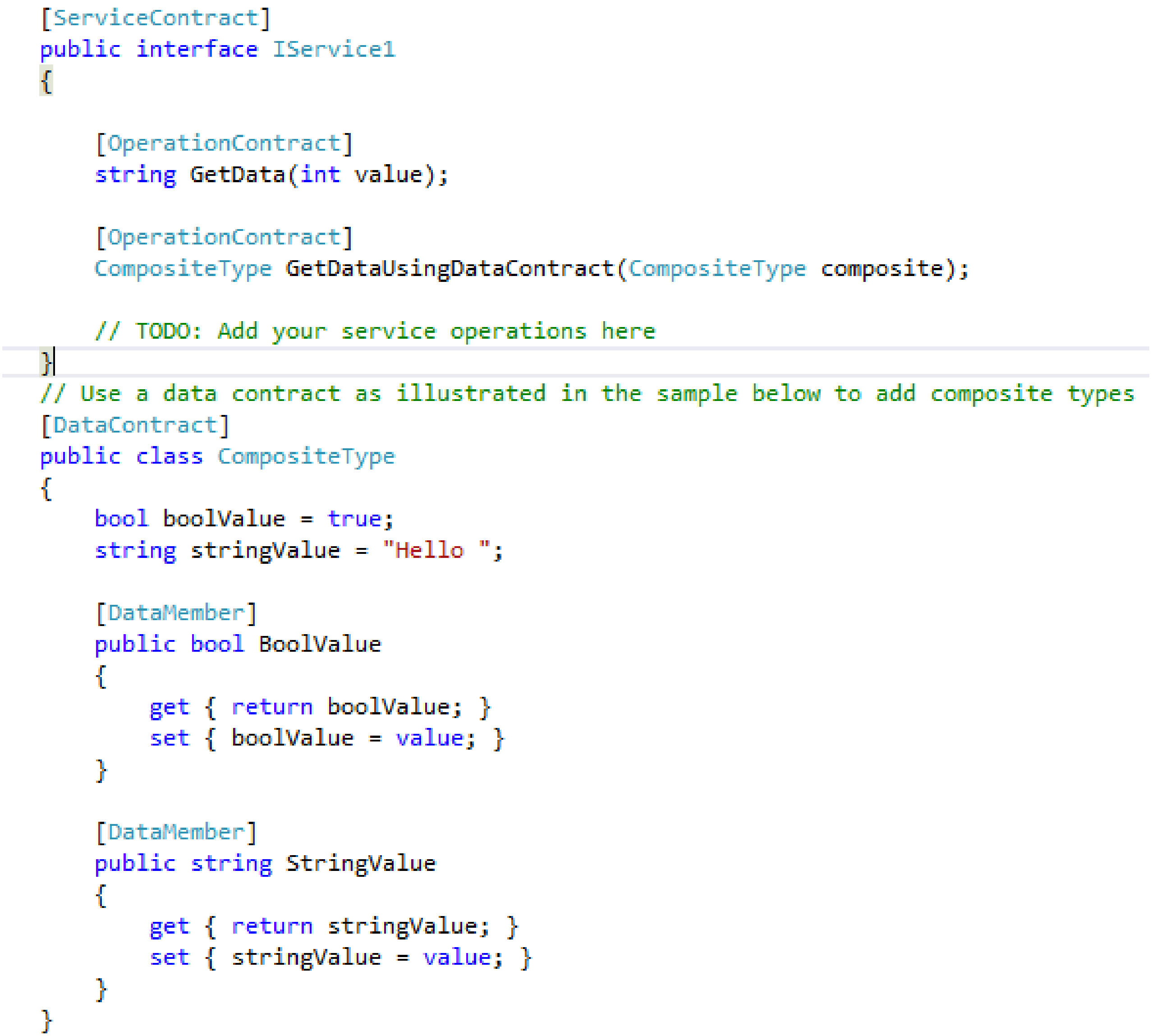
6.1 Publishing SOAP-Based Web Services

Step 2 Define the Service Contract

File **Iservice1.cs** defines file **IService1** **`interface`** and class **CompositeType** marked with a **DataContract** **attribute**.

class CompositeType contains **two** sample web service methods **GetData** and **GetDataUsingDataContract**

6.1 Publishing SOAP-Based Web Services



```
[ServiceContract]
public interface IService1
{

    [OperationContract]
    string GetData(int value);

    [OperationContract]
    CompositeType GetDataUsingDataContract(CompositeType composite);

    // TODO: Add your service operations here
}
// Use a data contract as illustrated in the sample below to add composite types
[DataContract]
public class CompositeType
{
    bool boolValue = true;
    string stringValue = "Hello ";

    [DataMember]
    public bool BoolValue
    {
        get { return boolValue; }
        set { boolValue = value; }
    }

    [DataMember]
    public string StringValue
    {
        get { return stringValue; }
        set { stringValue = value; }
    }
}
```

The contents of **file IService1.cs** created by default

6.1 Publishing SOAP-Based Web Services

Step 2 Define the Service Contract

The code- behind file **Service1** implements the **IService1** **interface.**

IService1 **interface** that **must** be **marked** by **ServiceContract attribute** and the web **service methods** **must** be **marked** by **OperationContract attribute** .

The **sample** web service **implements** a method **Welcome** that takes a **name** (represented as a **string**) as an argument and appends it to the welcome message that is **returned** (**string**) to the client.

6.1 Publishing SOAP-Based Web Services

Step 2 Define the Service Contract

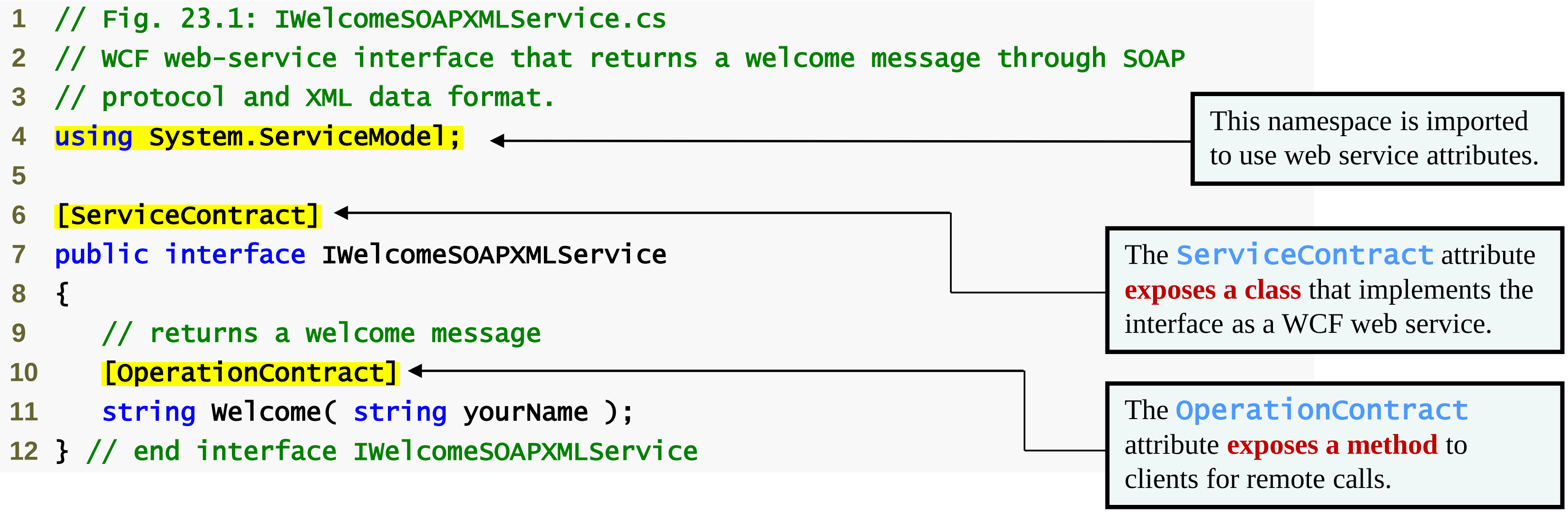
When creating services in Visual Web Developer, you **work** almost exclusively in the **code-behind files**. Accordingly, **modify** and **rename** the name and the contents of the **default code- behind file** as necessary.

For instance, the necessary changes in the sample application are provided in the following slide. **Leave** `Web.config` file **as it is** by **default**.

Start by defining the Service contract **interface** , renamed to **IWelcomeSOAPXMLService**, where the method **Welcome** is defined. Note, the required attribute **ServiceContract** for the **interface** and **OperationContract**, for each one of the **methods** in that interface

- Figure 15b.1 is a web service interface, which describes the methods and properties the client uses to access the service.

IWelcomeSOAPXML Service.cs



```
// Fig. 23.1: IWelcomeSOAPXMLService.cs
// WCF web-service interface that returns a welcome message through SOAP
// protocol and XML data format.
using System.ServiceModel;

[ServiceContract]
public interface IWelcomeSOAPXMLService
{
   // returns a welcome message
   [OperationContract]
   string Welcome( string yourName );
} // end interface IWelcomeSOAPXMLService
```

This namespace is imported to use web service attributes.

The `ServiceContract` attribute **exposes a class** that implements the interface as a WCF web service.

The `OperationContract` attribute **exposes a method** to clients for remote calls.

Fig. 15b.1 | WCF web-service interface that returns a welcome message through SOAP protocol and XML format.

6.1 Publishing SOAP-Based Web Services

The `ServiceContract` attribute **exposes a class that implements the interface** as a WCF web service.

The `OperationContract` attribute **exposes a method** to clients for remote calls.

6.1 Publishing SOAP-Based Web Services

Step 3 **Implement** the Web Service

Next, **write the required implementation** of this interface in the **code- behind** file, where the **file** and the **class** are renamed to **WelcomeSOAPXMLService.**

Outline

- Figure 15b.2 defines the class that implements the interface declared as the `ServiceContract`.

`WelcomeSOAPXML`
`Service.svc.cs`

```
// Fig. 23.2: WelcomeSOAPXMLService.svc.cs
// WCF web service that returns a welcome message using SOAP protocol and
// XML data format.
public class WelcomeSOAPXMLService : IWelcomeSOAPXMLService
{
   // returns a welcome message
   public string Welcome( string yourName )
   {
      return string.Format( "Welcome to WCF Web Services"
         + " with SOAP and XML, {0}!", yourName );
   } // end method Welcome
} // end class WelcomeSOAPXMLService
```

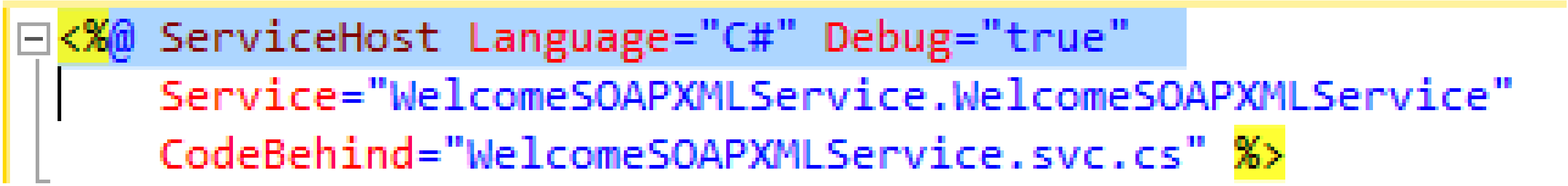
Implementing the `Welcome` method.

Fig. 15b.2 | WCF web service that returns a welcome message through the SOAP protocol and XML format.

6.1 Publishing SOAP-Based Web Services

Step 3 **Implement** the Web Service

Examine the markup of the SVC file



```
<%@ ServiceHost Language="C#" Debug="true"
    Service="WelcomeSOAPXMLService.WelcomeSOAPXMLService"
    CodeBehind="WelcomeSOAPXMLService.svc.cs" %>
```

6.1 Publishing SOAP-Based Web Services

Step 4 **Publish** and **Test** the web service

1. Build the solution

(**`Build-> Build WelcomeSOAPXMLService`**)

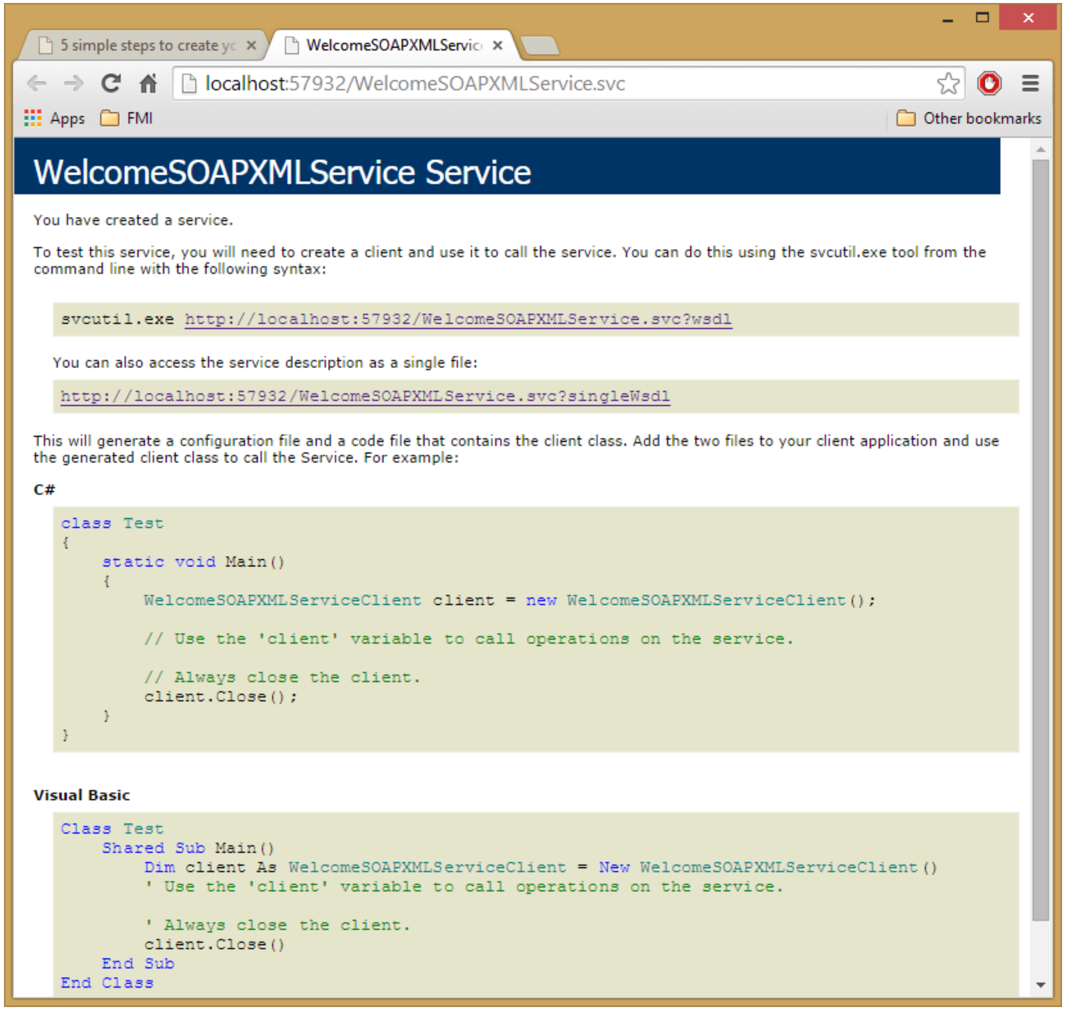
to ensure that the web service compiles without errors.

2. Right click the

`WelcomeSOAPXMLService.svc`

file and select **`View in Browser`** to **publish** the web service and test it using the **Local IIS** in Visual studio. Therefore, **the web service is running only while the solution is open in VS**.

6.1 Publishing SOAP-Based Web Services


5 simple steps to create yo
WelcomeSOAPXMLServic
localhost:57932/WelcomeSOAPXMLService.svc
Apps
FMI
Other bookmarks

WelcomeSOAPXMLService Service

You have created a service.

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

```
svcutil.exe http://localhost:57932/WelcomeSOAPXMLService.svc?wsdl
```

You can also access the service description as a single file:

```
http://localhost:57932/WelcomeSOAPXMLService.svc?singleWsdl
```

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

C#

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

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

        // Always close the client.
        client.Close();
    }
}
```

Visual Basic

```
Class Test
    Shared Sub Main()
        Dim client As WelcomeSOAPXMLServiceClient = New WelcomeSOAPXMLServiceClient()
        ' Use the 'client' variable to call operations on the service.

        ' Always close the client.
        client.Close()
    End Sub
End Class
```

The Service.svc file for the WelcomeSOAPXMLService WCF web service

6.1 Publishing SOAP-Based Web Services

Step 4 **Publish** and **Test** the web service

Access the **service information** from a browser using the link displayed in the browser

http://localhost:57932/WelcomeSOAPXMLService.svc

The web service description (**WSDL**) is found using the link

http://localhost:57932/WelcomeSOAPXMLService.svc?singleWsdl

6.1 Publishing SOAP-Based Web Services

A **service description** is an XML document that conforms to the **Web Service Description Language** (**WSDL**).

- **WSDL** is an XML vocabulary that defines the methods that a web service makes available.
- The WSDL document also **specifies lower-level information**.

When viewed in a web browser, an **SVC file presents a link to the service's WSDL document**.

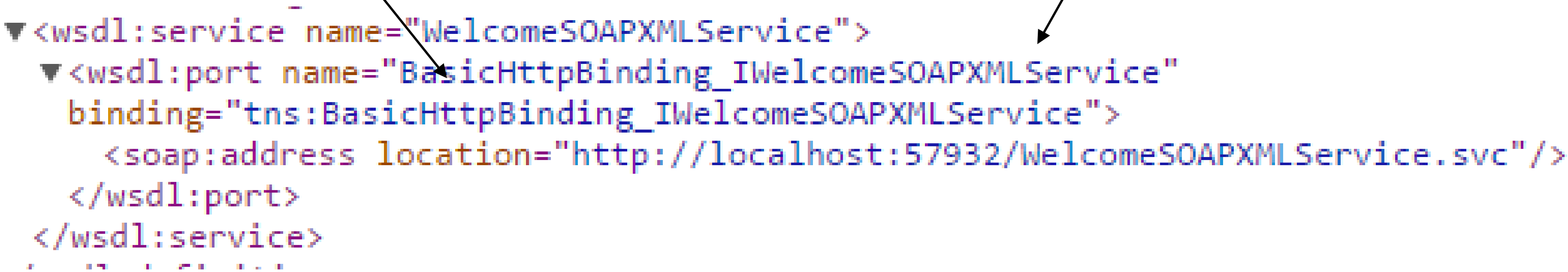
Copy the SVC URL (which ends with .svc) from the browser's address field as you'll need it to **discover** the **web service** when building the client application.

- The **WSDL** file specifies the service’s configuration information.
- Figure 15b.4 shows the `wsdl:service` element of the of the **`WSDL service file`** (found at the end of this file) .

`BasicHttpBinding` means that data is transferred over HTTP using XML.

Specifying the name of the service.

Specifying the service location.



```
▼<wsdl:service name="WelcomeSOAPXMLService">
  ▼<wsdl:port name="BasicHttpBinding_IWelcomeSOAPXMLService"
   binding="tns:BasicHttpBinding_IWelcomeSOAPXMLService">
     <soap:address location="http://localhost:57932/WelcomeSOAPXMLService.svc"/>
   </wsdl:port>
 </wsdl:service>
```

Fig. 15b.4 | Part of the **`WSDL service file`**
`http://localhost:57932/WelcomeSOAPXMLService.svc?singleWsdl`

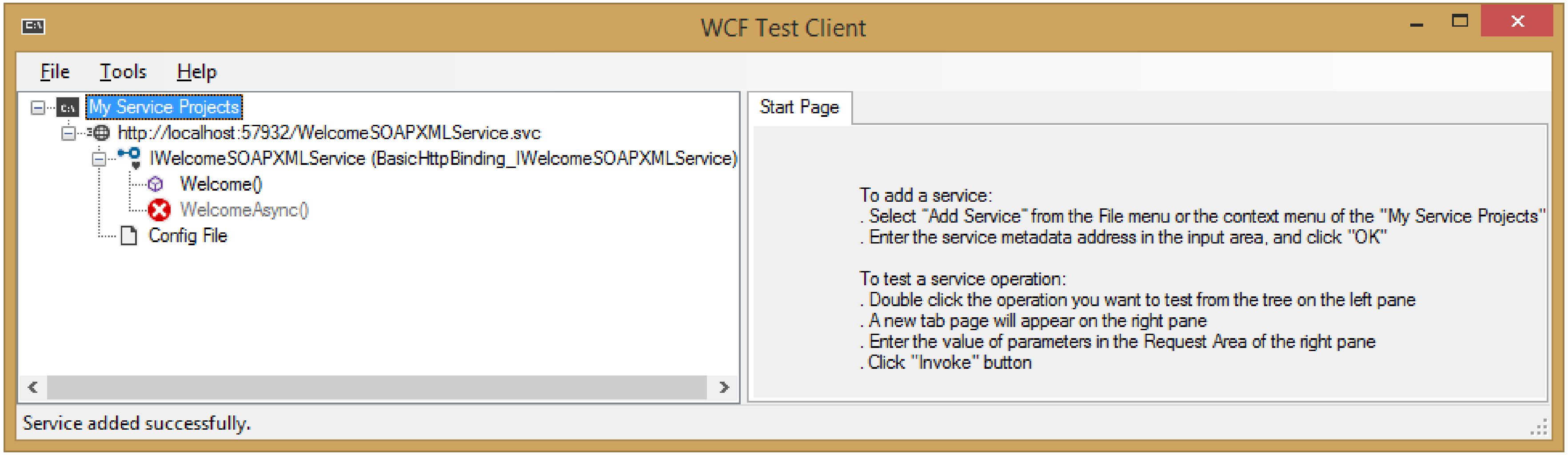
6.1 Publishing SOAP-Based Web Services

Step 4 Publish and **Test** the web service

Test the web service as follows:

Start without Debugging the **WCF Service Application** project

Double click the web service method to test it



WCF Test Client
File
Tools
Help
My Service Projects
http://localhost:57932/WelcomeSOAPXMLService.svc
IWelcomeSOAPXMLService (BasicHttpBinding_IWelcomeSOAPXMLService)
Welcome()
WelcomeAsync()
Config File
Start Page

To add a service:
. Select "Add Service" from the File menu or the context menu of the "My Service Projects"
. Enter the service metadata address in the input area, and click "OK"

To test a service operation:
. Double click the operation you want to test from the tree on the left pane
. A new tab page will appear on the right pane
. Enter the value of parameters in the Request Area of the right pane
. Click "Invoke" button

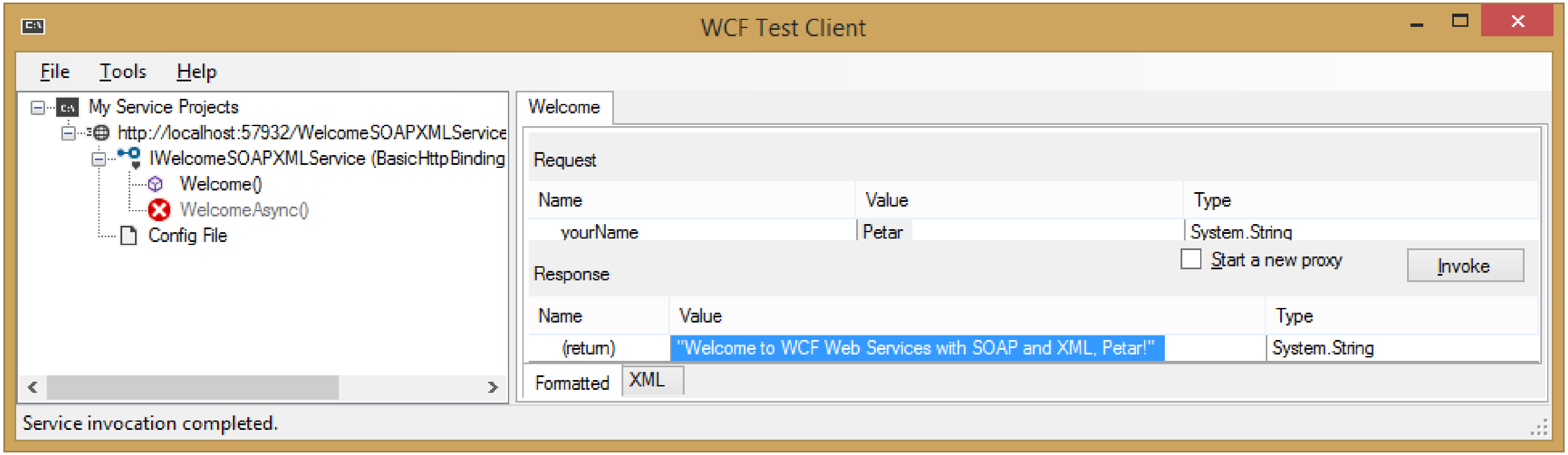
Service added successfully.

6.1 Publishing SOAP-Based Web Services

Step 4 **Publish** and **Test** the web service

Provide values for the arguments of the selected web method and click the button **Invoke** to test the method output

Leave this window open, if you deploy this service on IIS Express



Request

Name	Value	Type
yourName	Petar	System.String

Response

Name	Value	Type
(return)	"Welcome to WCF Web Services with SOAP and XML, Petar!"	System.String

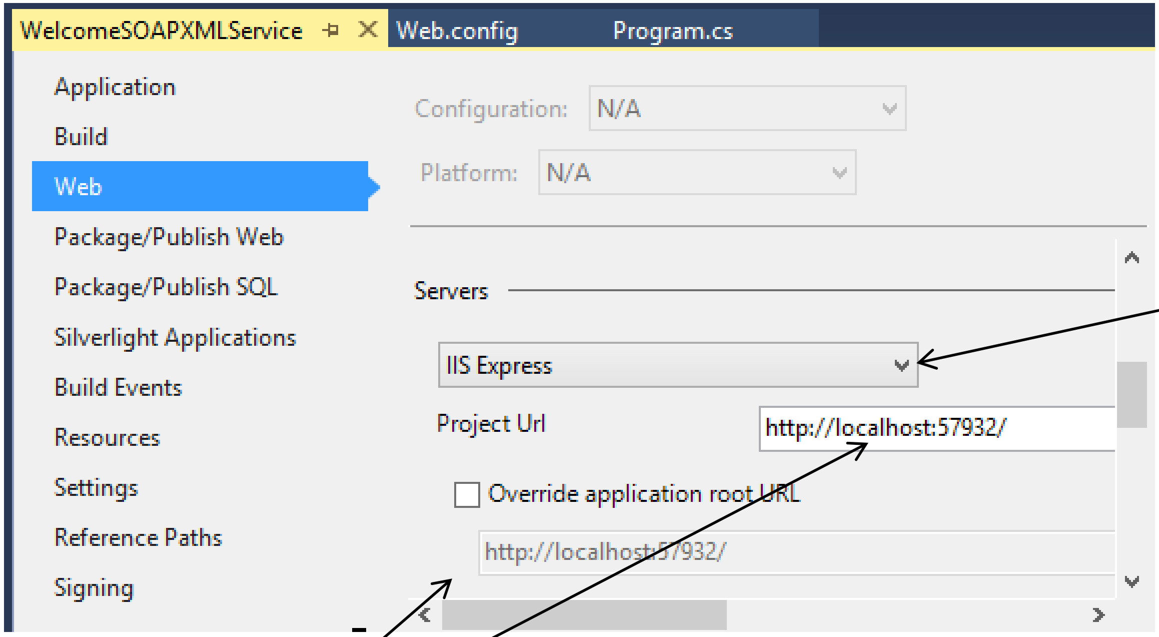
Service invocation completed.

6.1 Publishing SOAP-Based Web Services

By default, the ASP.NET Development Server **assigns a random port number** to each website it hosts.

Click on the **Project Properties** in the **Solution Explorer** and **adjust,** if necessary the port number used by the web service.

6.1 Publishing SOAP-Based Web Services



Configuration: N/A

Platform: N/A

Servers

IIS Express

Project Url http://localhost:57932/

☐ Override application root URL

http://localhost:57932/

Select the web server for deploying the web service. Here IIS is selected

Set the **Project Url ports property to Port number** that you want to use, which can be any unused TCP port

WCF service Application **Properties** window

6.2 Consuming SOAP-Based Web Services

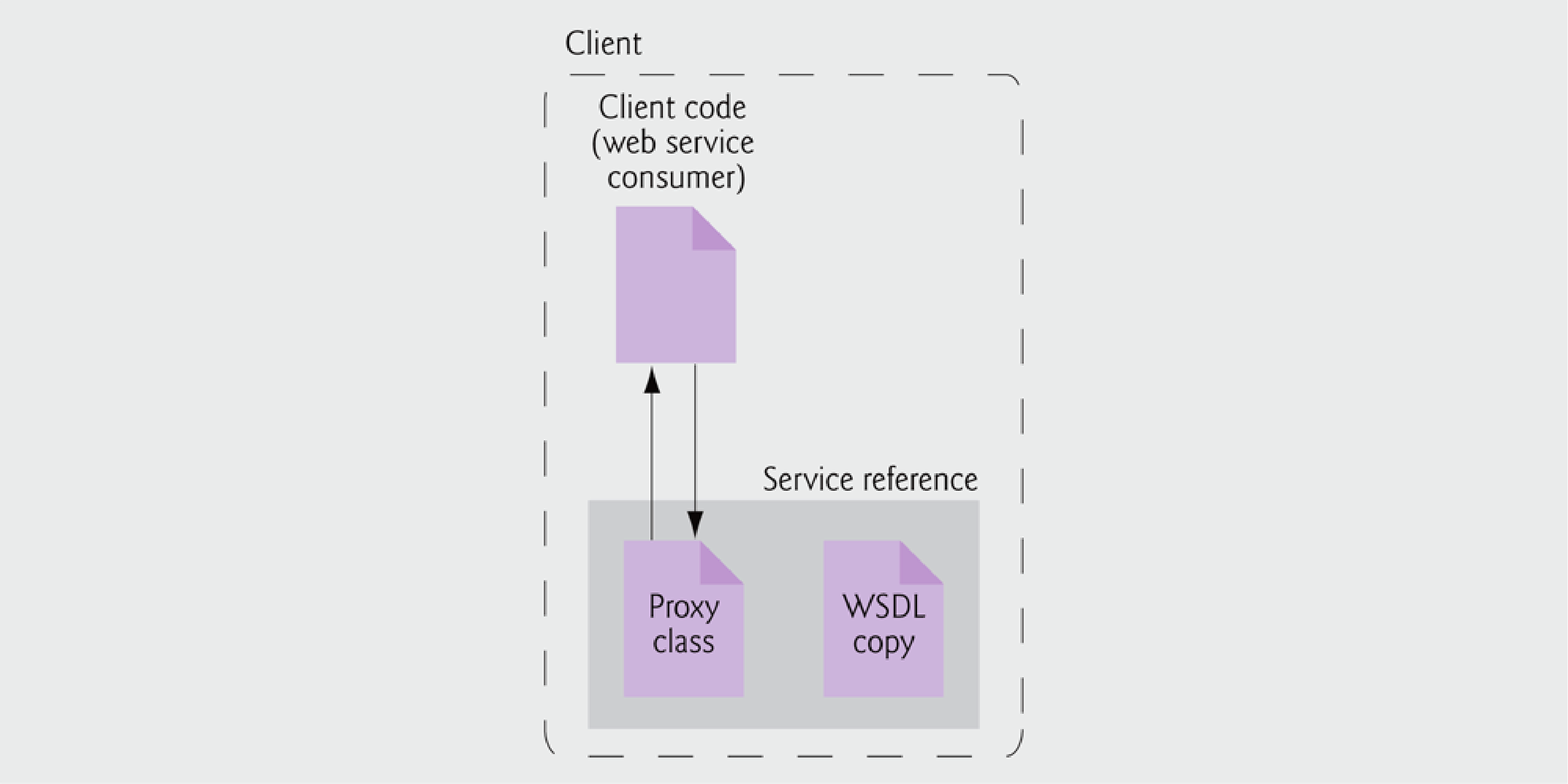
Creating a Client to Consume the **WelcomeSOAPXMLService** is rendered just to **adding a service reference** to the client and **creating an of a proxy class** that represents the web service in the service reference.

The **client application accesses** the web service via an instance of the **proxy class**.

Note: You can consume the same way **freely available web services** published, for instance at

Directory of Public SOAP Web Services

6.2 Consuming SOAP-Based Web Services



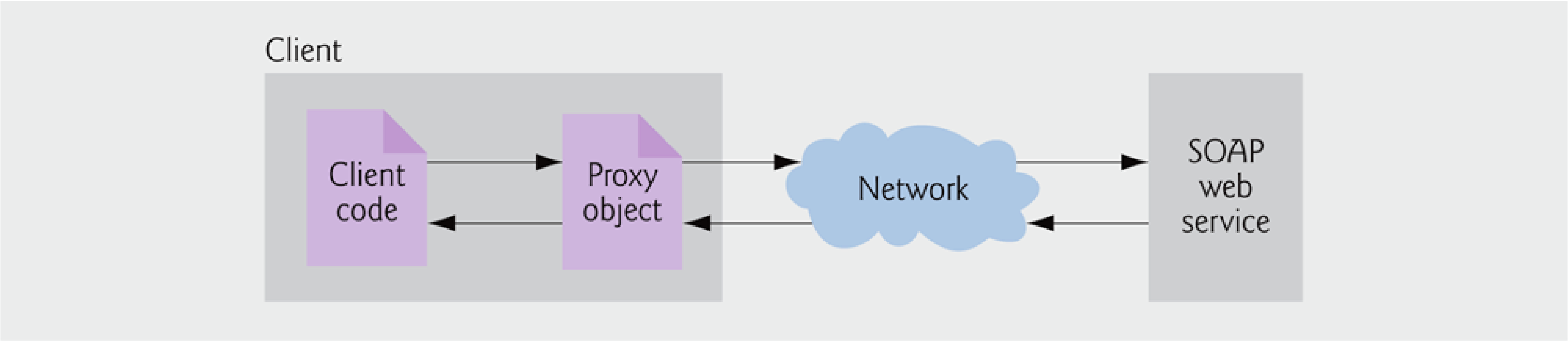
NET WCF web service client after a web-service reference has been added.

6.2 Consuming SOAP-Based Web Services

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.

6.2 Consuming SOAP-Based Web Services



Interaction between a web-service client and a SOAP web service.

6.2 Consuming SOAP-Based Web Services

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. The following slide depicts the interactions among the client code, proxy class and web service.

The proxy class (**`Reference.cs`**) is not shown in the **Solution Explorer**, unless the button **`Show All files`** is clicked in the **Solution Explorer**

6.2 Consuming SOAP-Based Web Services

Create an application in Visual C# 2017 named `WelcomeSOAPXMLClient`

In case the web service is deployed on **IIS Express (integrated in VS)** then create `WelcomeSOAPXMLClient` *in the same solution* of the web service or keep the window for testing the web service open (**The web service must be running while discovering it**)

For **deploying** **the web service** on the **local IIS** or **an external host** follow the attached tutorial `DeployWebServiceIMG.pdf`

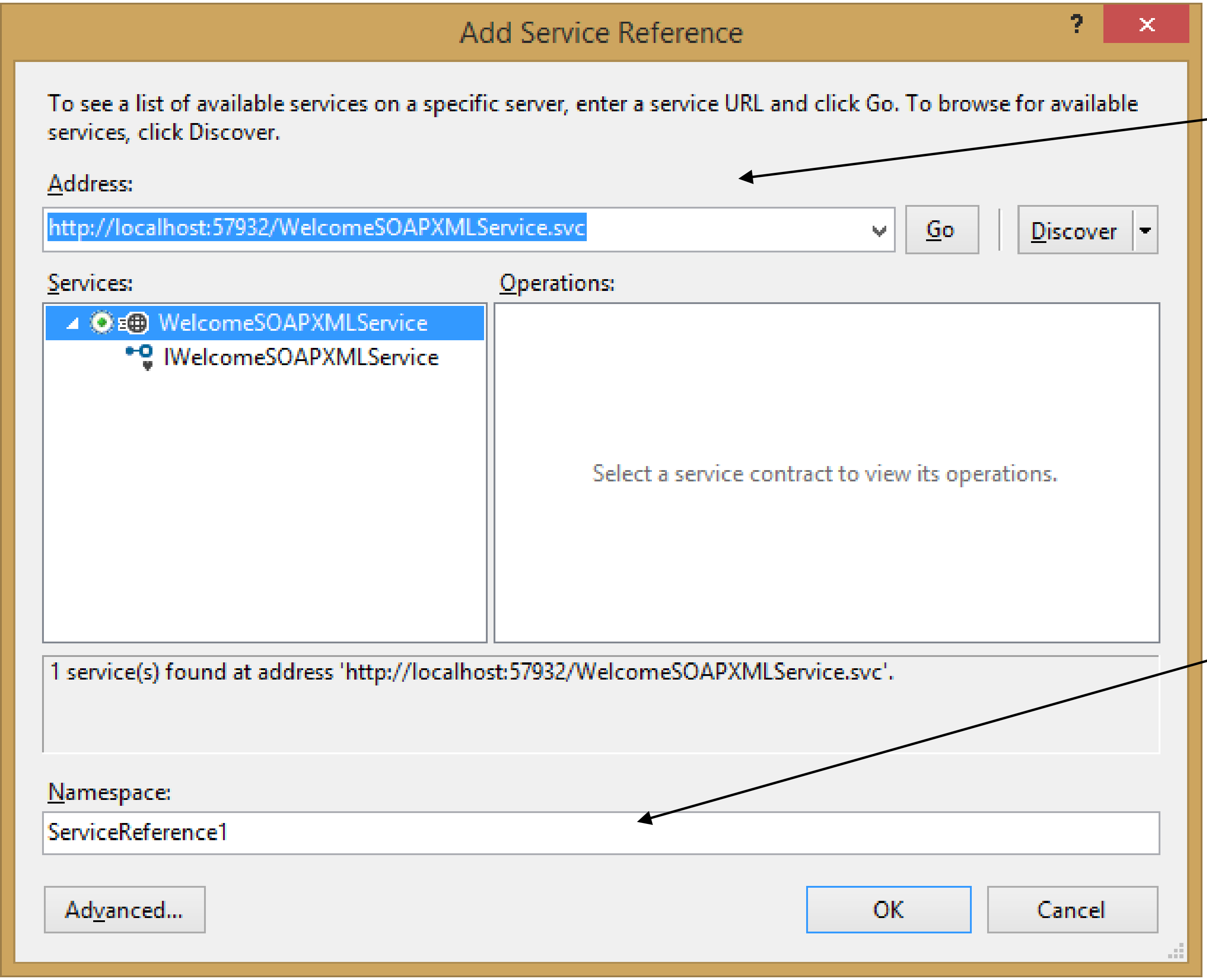
6.2 Consuming SOAP-Based Web Services

Right click the project name in the **Solution Explorer** and select **Add Service Reference…**

- **Enter the URL** of `WelcomeSOAPXMLService`’s `.svc` file in the **Address** field.
- Change the **Namespace** field to `ServiceReference`.

Click the **Ok** button. The **Solution Explorer** should now contain a **Service References** folder.

6.2 Consuming SOAP-Based Web Services



Add Service Reference

To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

Address: http://localhost:57932/WelcomeSOAPXMLService.svc

Services:
- WelcomeSOAPXMLService
  - IWelcomeSOAPXMLService

Operations:

Select a service contract to view its operations.

1 service(s) found at address 'http://localhost:57932/WelcomeSOAPXMLService.svc'.

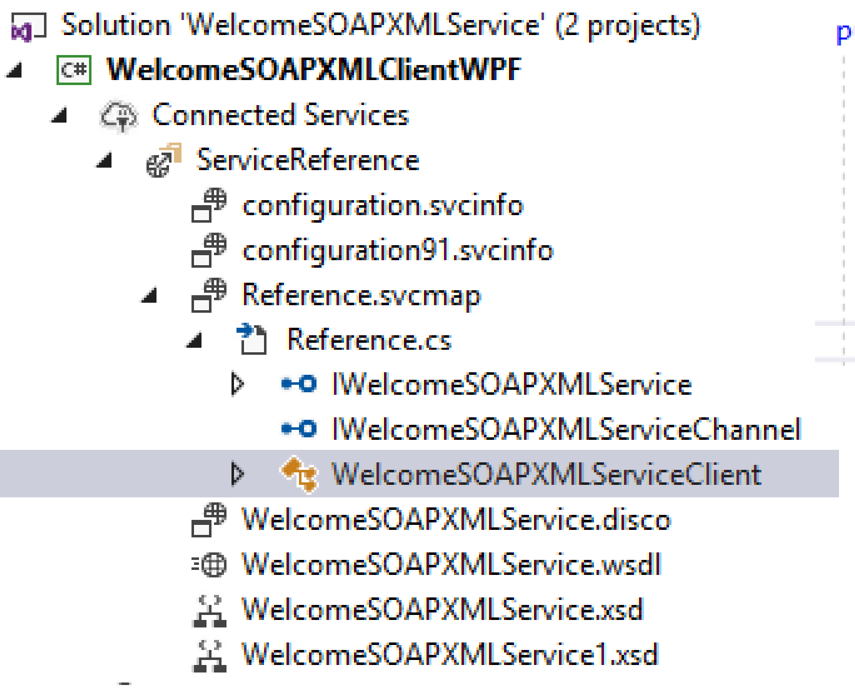
Namespace: ServiceReference1

Discover a Web service to consume by a client

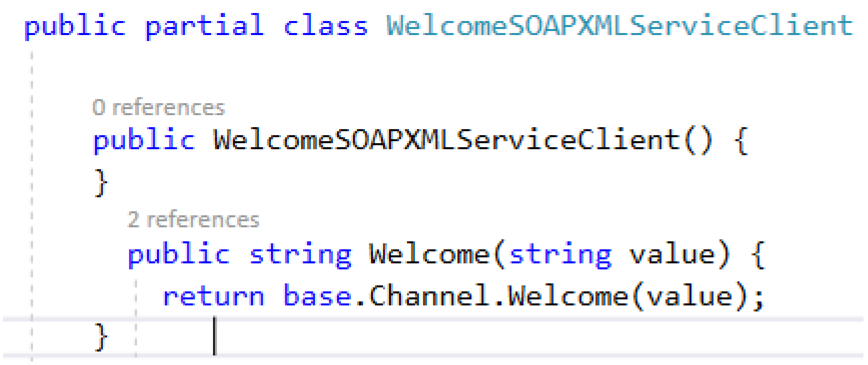
Defines the namespace where the proxy class will be created

6.2 Consuming SOAP-Based Web Services

Update the **namespace** in the textbox, for example, to **`ServiceReference`**. It defines the namespace **`WelcomeSOAPXMLClientWPF.ServiceReference`** where the **proxy class** named as **`WelcomeSOAPXMLServiceClient`** will be created.



```
Solution 'WelcomeSOAPXMLService' (2 projects)
◢ WelcomeSOAPXMLClientWPF
    ◢ Connected Services
        ◢ ServiceReference
            configuration.svcinfo
            configuration91.svcinfo
          ◢ Reference.svcmap
              ◢ Reference.cs
                  ▷ IWelcomeSOAPXMLService
                    IWelcomeSOAPXMLServiceChannel
                  ▷ WelcomeSOAPXMLServiceClient
            WelcomeSOAPXMLService.disco
            WelcomeSOAPXMLService.wsdl
            WelcomeSOAPXMLService.xsd
            WelcomeSOAPXMLService1.xsd
```



```
public partial class WelcomeSOAPXMLServiceClient

    0 references
    public WelcomeSOAPXMLServiceClient() {
    }
      2 references
      public string Welcome(string value) {
        return base.Channel.Welcome(value);
    }
```

Outline

WelcomeSOAPXMLForm.cs

(1 of 2)

- The application in Fig. 15b.10 uses the `WelcomeSOAPXMLService` service to send a welcome message.

```
// WelcomeSOAPXMLForm.cs
// Client that consumes WelcomeSOAPXMLService.
using System;
using System.Windows;

namespace WelcomeSOAPXMLClient
{
    public partial class WelcomeSOAPXML : Window
    {
        // declare a reference to web service
        private ServiceReference.WelcomeSOAPXMLServiceClient client;

        public WelcomeSOAPXML()
        {
            InitializeComponent();
            client = new ServiceReference.WelcomeSOAPXMLServiceClient();
        } // end constructor
```

Declaring the web service's proxy object.

Creating the proxy object.

Fig. 15b.10 | Client that consumes `WelcomeSOAPXMLService`. (Part 1 of 2.)

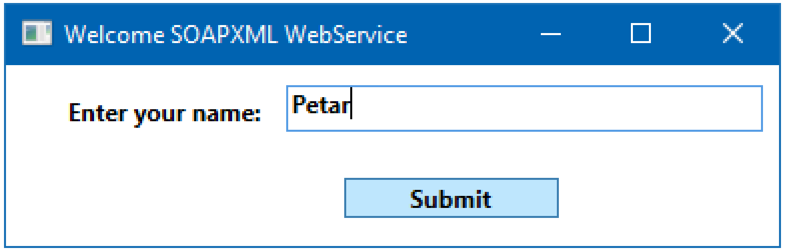
```

      // creates welcome message from text input and web service
      private void BtnSubmit_Click( object sender, RoutedEventArgs e )
      {
         MessageBox.Show( client.Welcome( TxtYourName.Text ), "Welcome" );
      } // end method submitButton_Click
   } // end class WelcomeSOAPXML window
} // end namespace WelcomeSOAPXMLClient
```

WelcomeSOAPXMLForm.cs

(2 of 2)

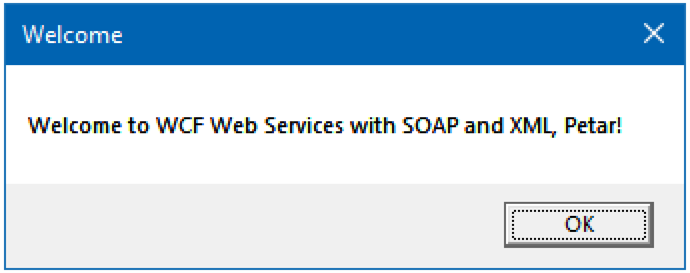
Invoking the web service's `Welcome` method.

a) User inputs name.



Enter your name: Petar

b) Message sent from WelcomeSOAPXMLService.



Welcome to WCF Web Services with SOAP and XML, Petar!

Fig. 15b.10 | Client that consumes `WelcomeSOAPXMLService`. (Part 2 of 2.)

6.2 Consuming SOAP-Based Web Services

Note: Remember always to close the client of the web service as

```
((IDisposable)client).Dispose();
```

6.2 Consuming SOAP-Based Web Services

```
//Done with the service, let's close it.
try
{
    if (client.State !=
      System.ServiceModel.CommunicationState.Faulted)
    {
       client.Close();
    }
}
catch (Exception ex)
{
     client.Abort();
}
```

6.3 Using Data Contracts

A ***data contract*** is a formal agreement between a service and a client that abstractly describes the data to be exchanged.

That is, to communicate, the client and the service do not have to share the same types, only the same data contracts.

A data contract precisely defines, for each parameter or return type, what data is serialized (turned into XML) to be exchanged

6.3 Using Data Contracts

Windows Communication Foundation (WCF) uses a **serialization engine** called the **Data Contract Serializer** by default to **serialize and deserialize** data (convert it to and from XML). All .NET Framework **primitive types, such as integers and strings**, as well as certain **types treated as primitives**, such as **DateTime** and **XmlElement**, can be **serialized** with no other preparation and are considered as **having default data contracts**. Many .NET Framework types also have existing data contracts

6.3 Using Data Contracts

New complex types that you create **must have a data contract** defined for them **to be serializable**. By default, the **`DataContractSerializer`** infers the data contract and **serializes all publicly visible types**. **All `public` read/write properties and fields of the type are serialized**.

You can **opt out members from serialization** by using the **`IgnoreDataMember Attribute`**.

6.3 Using Data Contracts

You can also **explicitly create a data contract** by using the **DataContract** and **DataMember** attributes. This is normally done by **applying** the **DataContract** attribute **to the type**. This attribute can be applied to classes, structures, and enumerations.

The **DataMember** attribute **must then be applied to each member of the data contract type** to indicate that it is a *data member*, that is**, it should be serialized**

6.3 Using Data Contracts

```
[ServiceContract]
public interface ISampleInterface
{
    // No data contract is requred since both the parameter
    // and return types are primitive types.
    [OperationContract]
    double SquareRoot(int root);

    // No Data Contract required because both parameter and return
    // types are marked with the SerializableAttribute attribute.
    [OperationContract]
    System.Drawing.Bitmap GetPicture(System.Uri pictureUri);

    // The MyTypes.PurchaseOrder is a complex type, and thus
    // requires a data contract.
    [OperationContract]
    bool ApprovePurchaseOrder(MyTypes.PurchaseOrder po);
}
```

6.3 Using Data Contracts

```
namespace MyTypes
{
    [DataContract]
    public class PurchaseOrder
    {
        private int poId_value;

        // Apply the DataMember Attribute to the property.
        [DataMember]
        public int PurchaseOrderId
        {

            get { return poId_value; }
            set { poId_value = value; }
        }
    }
}
```

6.3 Using Data Contracts

The following notes provide items to consider when creating data contracts:

- ✓The **IgnoreDataMemberAttribute** attribute is **only honored when used with unmarked types**. This includes types that are not marked with one of the attributes- **DataContract, Serializable**, **CollectionData, Contract**, or **EnumMember**, or marked as serializable by any other means (such as **IXmlSerializable**).

6.3 Using Data Contracts

You can **apply** the **DataMember** attribute to **fields**, and **properties**.

Member accessibility levels (internal, private, protected, or public) do not affect the data contract in any way.

The **DataMember** attribute is **ignored if it is applied to static members**.

During serialization, **get** property code **is called** for property data members to **get the value of the properties** to be serialized.

REST-Based XML Web Service

RESTful services are those which follow the REST (Representational State Transfer) architectural style. Before implementing your first RESTful service, lets first understand the concept behind it. As we know that WCF allows us to make calls and exchange messages using SOAP over a variety of protocols i.e. HTTP, TCP, Named Pipes and MSMQ etc. In a scenario, if we are using SOAP over HTTP, we are just utilizing HTTP as a transport. But HTTP is much more than just a transport. So, when we talk about REST architectural style, it dictates that "**Instead of using complex mechanisms like CORBA, RPC or SOAP for communication, simply HTTP should be used for making calls**"

http://www.topwcftutorials.net/2018/02/practical-wcf-restful-service.html

7 REST-Based XML Web Service

Following are 5 simple steps to **develop a RESTFul web service**.

Create a **WCF Service Application** Project.

Define the **Service Contract** and **Data Contracts** for user- defined types of parameters and return results of web service methods

Implement the Web Service

Publish and **Test** the web service

http://www.topwcftutorials.net/2013/09/simple-steps-for-restful-service.html

Alternative ways:

REST Model - Build a REST Service in Visual Studio 2015 Part 1

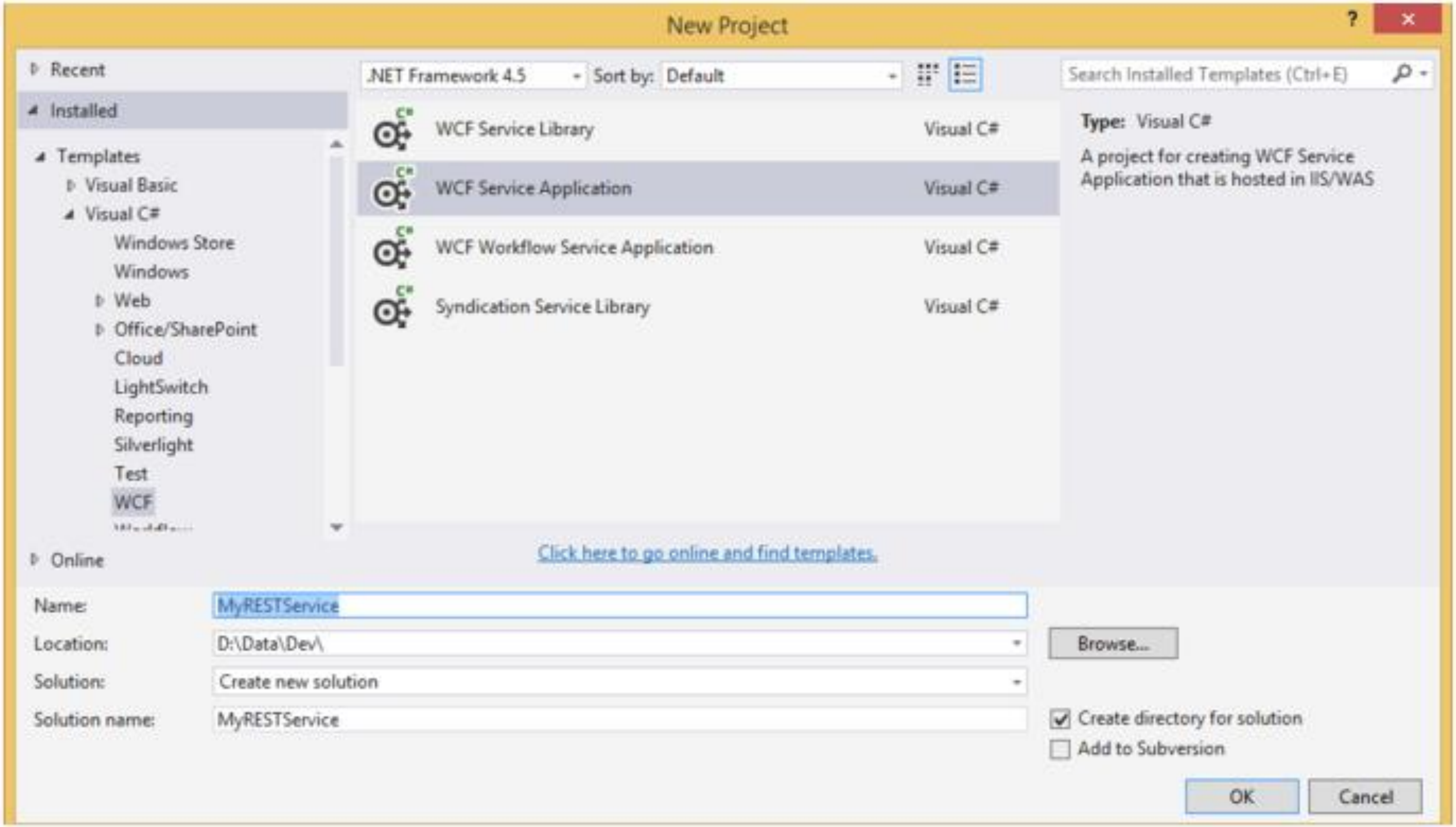
REST Model - Build a REST Service in Visual Studio 2015 Part 2

7 REST-Based XML Web Service

1. Open Visual Studio.

From **File -> New Project**. Select WCF from left and create a new **WCF Service Application**.

7 REST-Based XML Web Service



New Project
?
×
▷ Recent
◢ Installed
◢ Templates
▷ Visual Basic
◢ Visual C#
Windows Store
Windows
▷ Web
▷ Office/SharePoint
Cloud
LightSwitch
Reporting
Silverlight
Test
WCF
▷ Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (Ctrl+E)
WCF Service Library — Visual C#
WCF Service Application — Visual C#
WCF Workflow Service Application — Visual C#
Syndication Service Library — Visual C#
Click here to go online and find templates.
Browse...
OK
Cancel

Type: Visual C#

A project for creating WCF Service Application that is hosted in IIS/WAS

Name: MyRESTService

Location: D:\Data\Dev\

Solution: Create new solution

Solution name: MyRESTService

☑ Create directory for solution

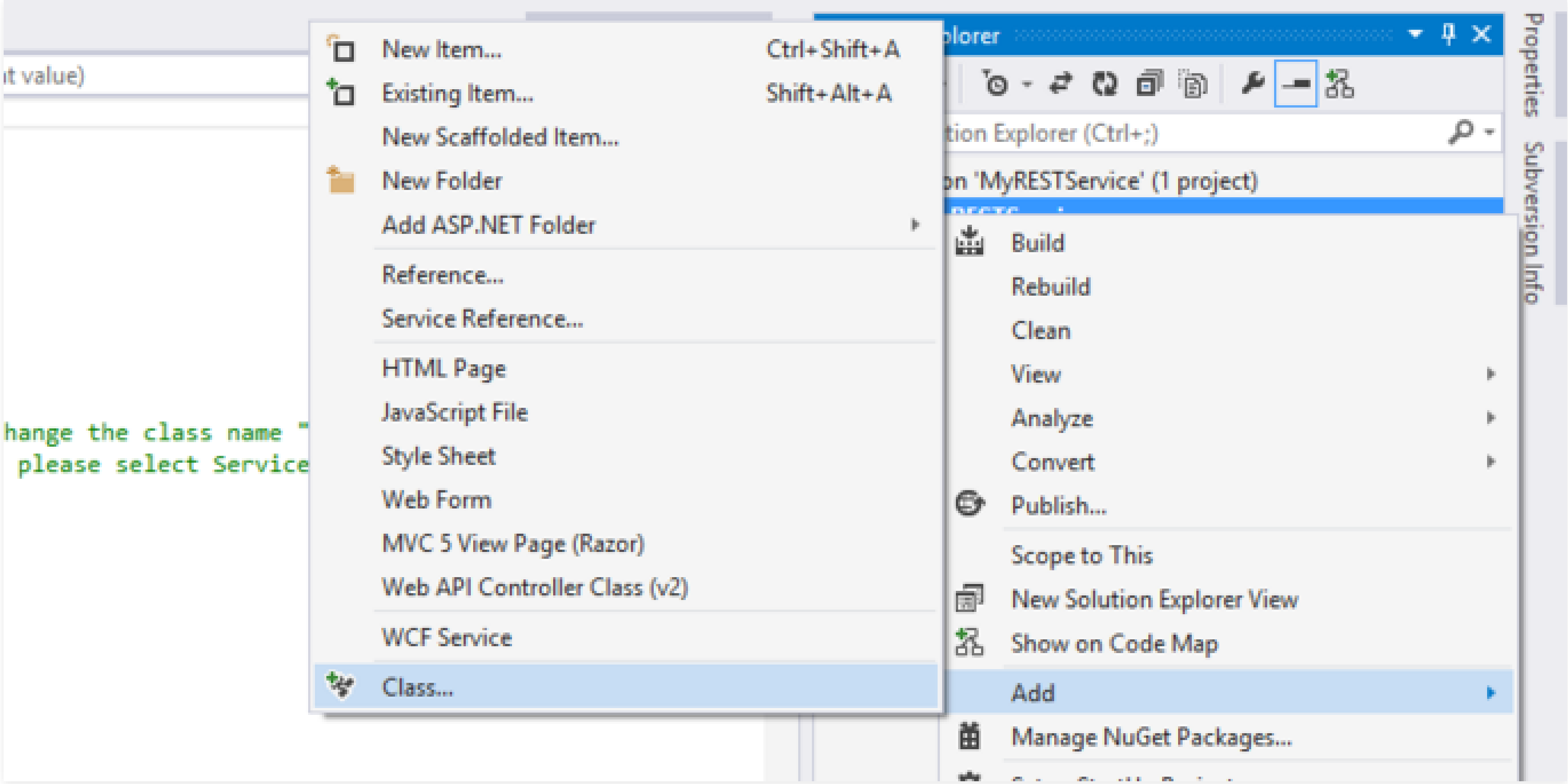
□ Add to Subversion

7 REST-Based XML Web Service

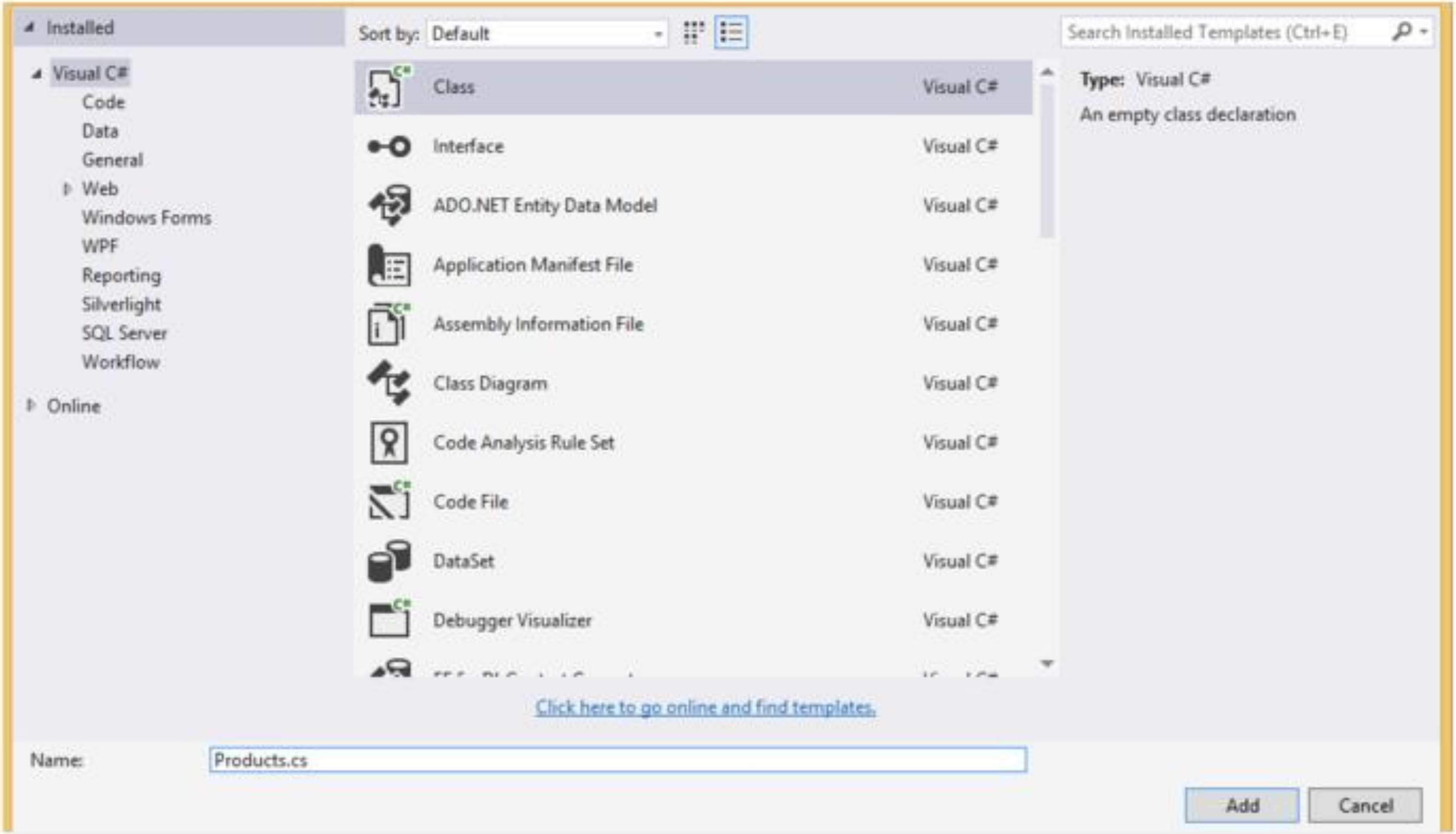
2. Preparing the data to return

Add a class to newly created project. Name it to Products.cs.

7 REST-Based XML Web Service



7 REST-Based XML Web Service

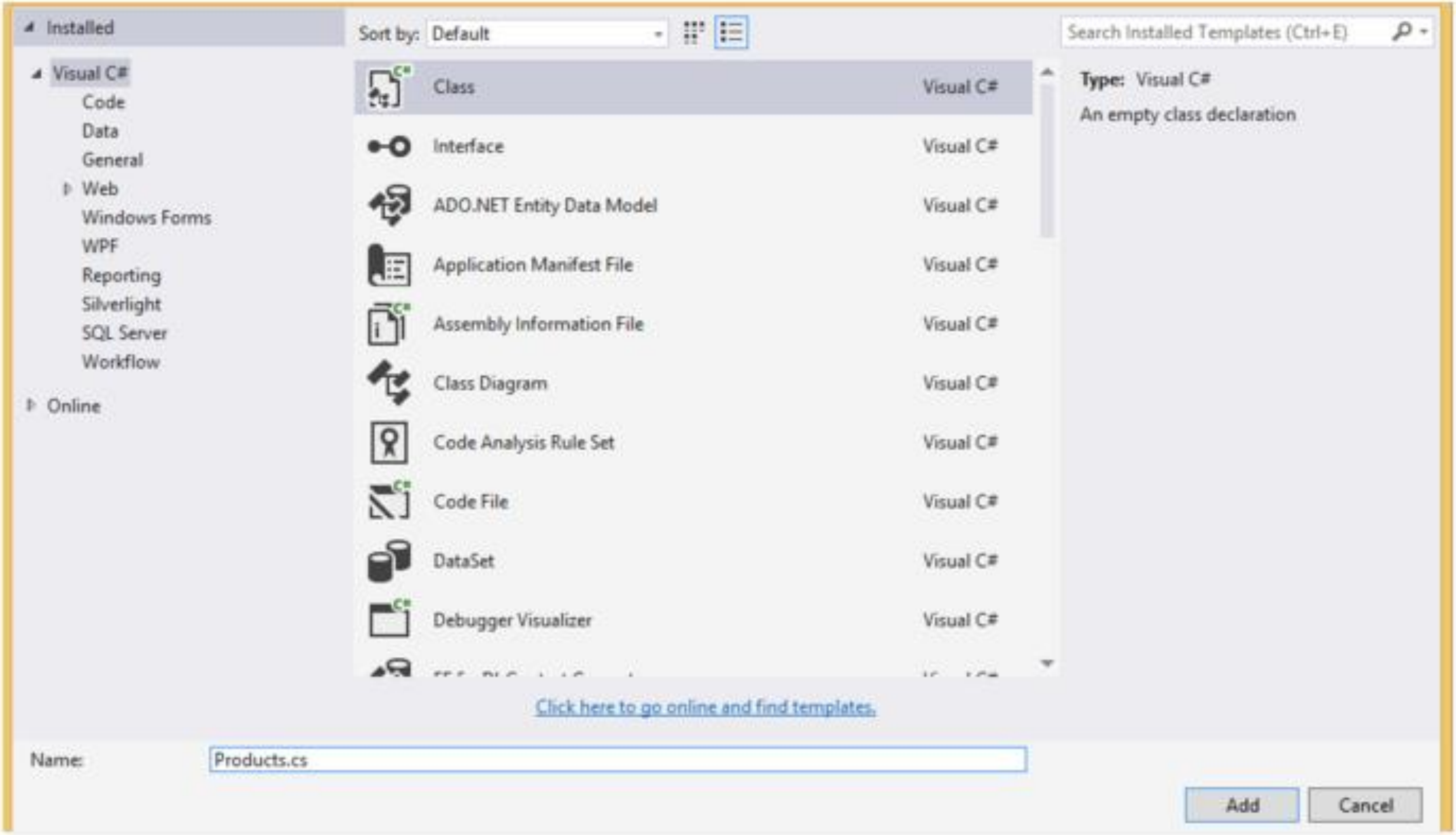

Installed
Visual C#
Code
Data
General
Web
Windows Forms
WPF
Reporting
Silverlight
SQL Server
Workflow
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Class Visual C#
Interface Visual C#
ADO.NET Entity Data Model Visual C#
Application Manifest File Visual C#
Assembly Information File Visual C#
Class Diagram Visual C#
Code Analysis Rule Set Visual C#
Code File Visual C#
DataSet Visual C#
Debugger Visualizer Visual C#
Click here to go online and find templates.
Add
Cancel

Type: Visual C#

An empty class declaration

Name: Products.cs

7 REST-Based XML Web Service



Sort by: Default

Type: Visual C#

An empty class declaration

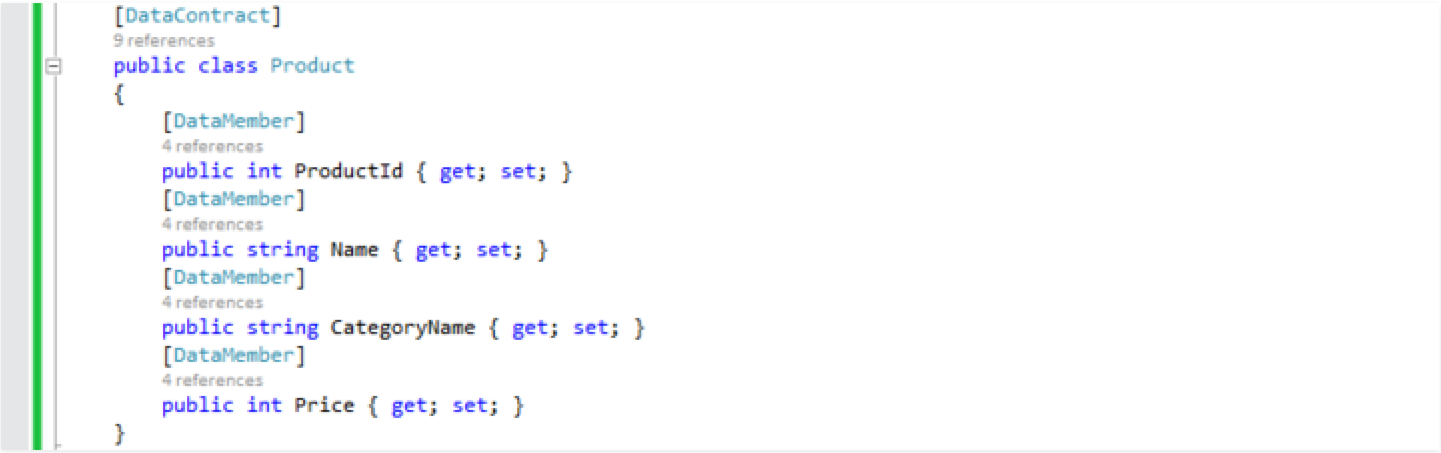
Name: Products.cs

7 REST-Based XML Web Service

Now this **`Products.cs`** file will contain two things. The first one is **`Data Contract`** as follows:

Don’t forget

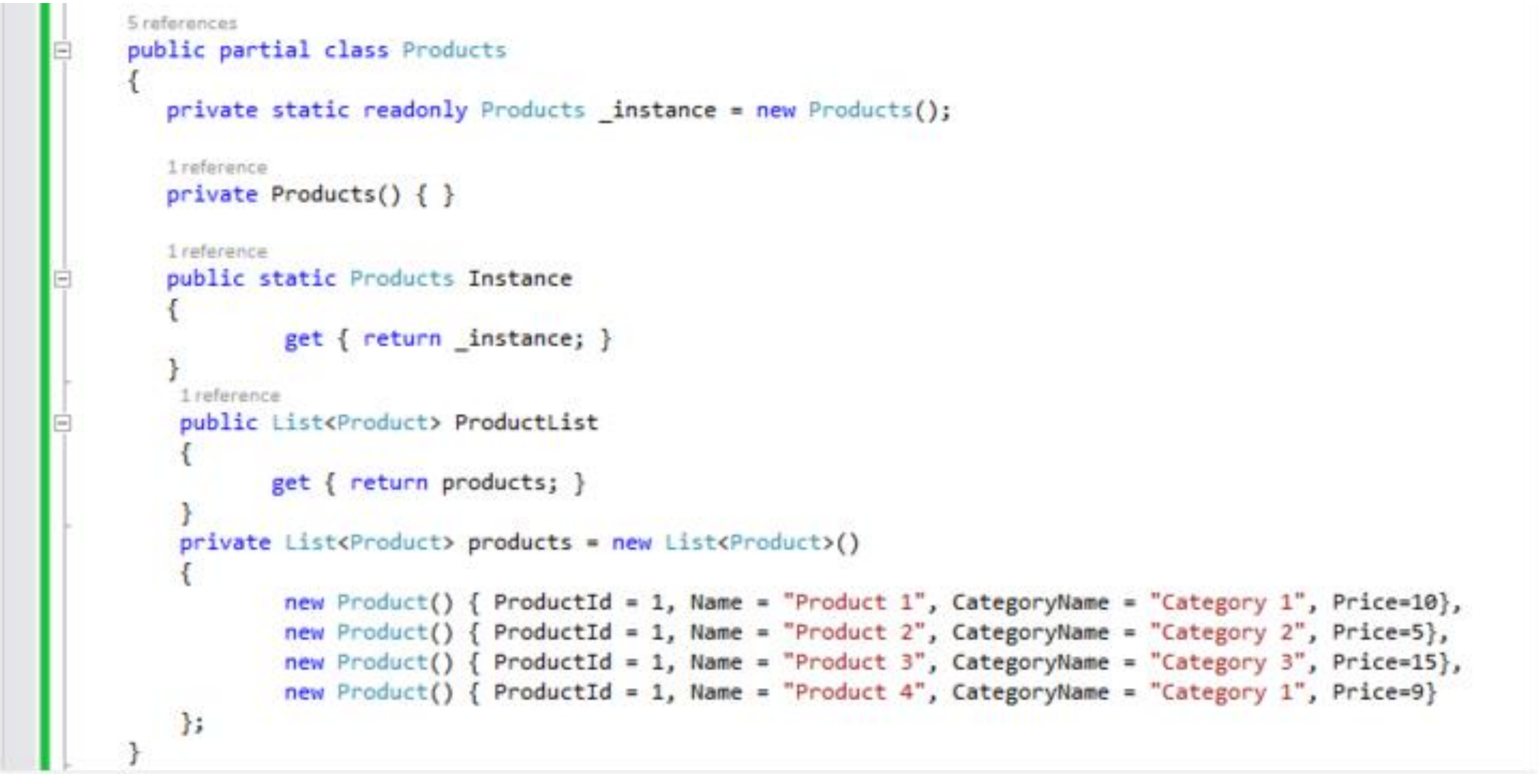
```
using System.ServiceModel;
using System.Runtime.Serialization;
```



```
[DataContract]
9 references
public class Product
{
    [DataMember]
    4 references
    public int ProductId { get; set; }
    [DataMember]
    4 references
    public string Name { get; set; }
    [DataMember]
    4 references
    public string CategoryName { get; set; }
    [DataMember]
    4 references
    public int Price { get; set; }
}
```

7 REST-Based XML Web Service

The second one is a singleton implemented **`class Products`** that gets products data from a database and return list of products, create an **`Instance`** and lookup its **`ProductList`**. For simplicity, we are preparing data inside this class instead of fetching from the database

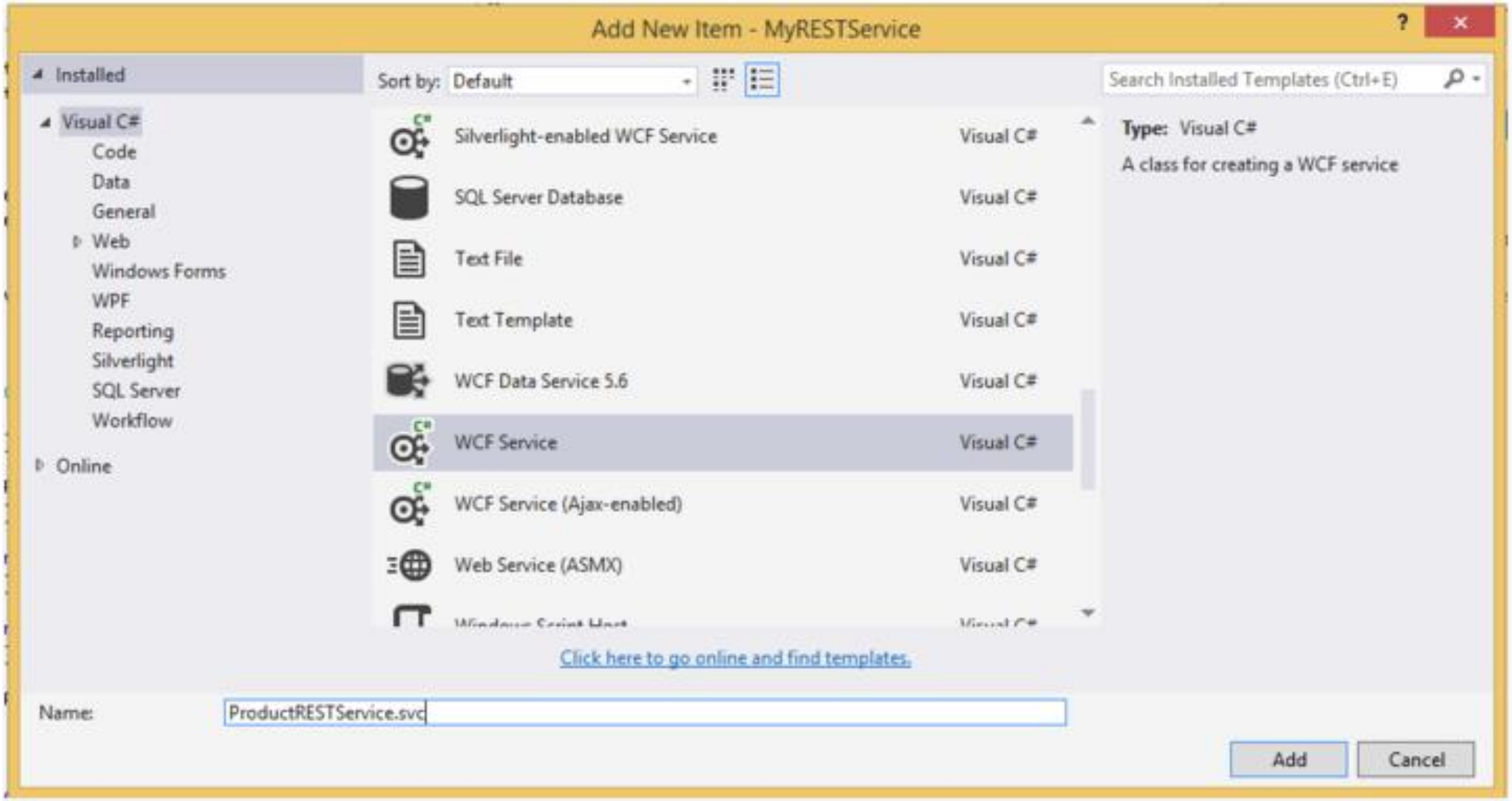


```
5 references
public partial class Products
{
    private static readonly Products _instance = new Products();

    1 reference
    private Products() { }

    1 reference
    public static Products Instance
    {
            get { return _instance; }
    }
     1 reference
     public List<Product> ProductList
     {
           get { return products; }
     }
     private List<Product> products = new List<Product>()
     {
            new Product() { ProductId = 1, Name = "Product 1", CategoryName = "Category 1", Price=10},
            new Product() { ProductId = 1, Name = "Product 2", CategoryName = "Category 2", Price=5},
            new Product() { ProductId = 1, Name = "Product 3", CategoryName = "Category 3", Price=15},
            new Product() { ProductId = 1, Name = "Product 4", CategoryName = "Category 1", Price=9}
     };
}
```

7 REST-Based XML Web Service

3. Creating Service Contract

Now add a new **WCF Service** to this project as follows:



Add New Item - MyRESTService

Installed

Sort by: Default

Search Installed Templates (Ctrl+E)

- Visual C#
  - Code
  - Data
  - General
  - Web
  - Windows Forms
  - WPF
  - Reporting
  - Silverlight
  - SQL Server
  - Workflow
- Online

Silverlight-enabled WCF Service	Visual C#
SQL Server Database	Visual C#
Text File	Visual C#
Text Template	Visual C#
WCF Data Service 5.6	Visual C#
WCF Service	Visual C#
WCF Service (Ajax-enabled)	Visual C#
Web Service (ASMX)	Visual C#
Windows Script Host	Visual C#

Type: Visual C#

A class for creating a WCF service

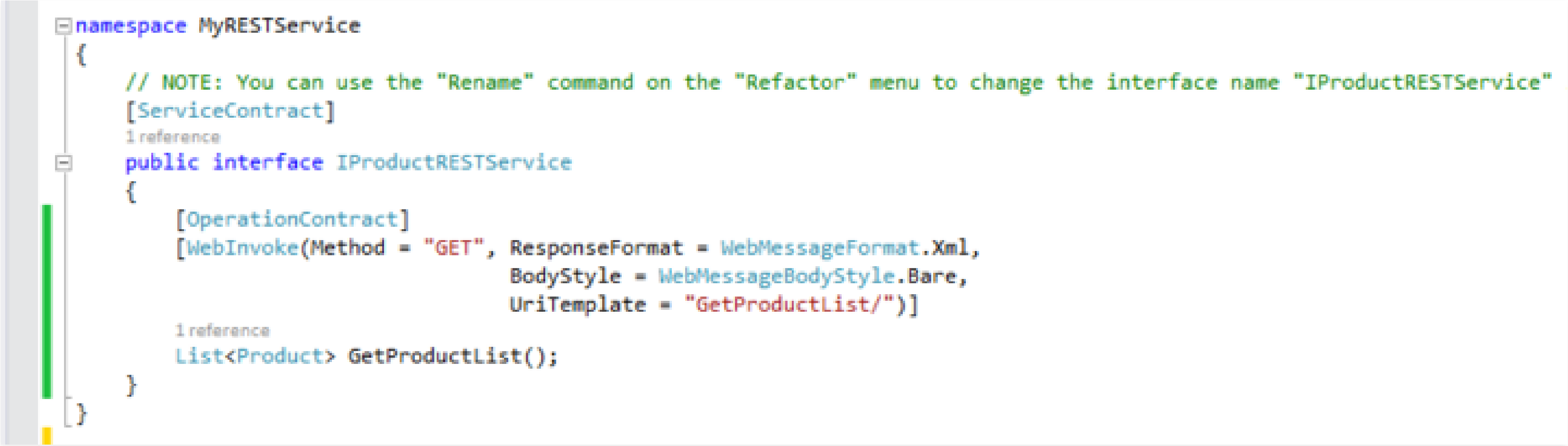
Click here to go online and find templates.

Name: ProductRESTService.svc

7 REST-Based XML Web Service

It will add contract as well as service file to project. Following is the code for **service contract** i.e. IProductRESTService.cs.

(A web service is added by default to a WCF Application project. Use the "Rename" command on the "Refactor" menu to change the class name "IProductRESTService")



```
namespace MyRESTService
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to change the interface name "IProductRESTService"
    [ServiceContract]
    1 reference
    public interface IProductRESTService
    {
        [OperationContract]
        [WebInvoke(Method = "GET", ResponseFormat = WebMessageFormat.Xml,
                                   BodyStyle = WebMessageBodyStyle.Bare,
                                   UriTemplate = "GetProductList/")]
        1 reference
        List<Product> GetProductList();
    }
}
```

7 REST-Based XML Web Service

IProductRESTService contains only one method i.e. GetProductList. Important points to understand about this method is WebInvoke attribute parameters.

Method = “GET”, represents an HTTP GET request.

ResponseFormat = WebMessageFormat.Xml, response format will be XML here but we can return JSON as well by changing its value to WebMessageFormat.json.

BodyStyle = **WebMessageBodyStyle.Bare**, indicates **neither the request and nor response are wrapped**. Other possible values for BodyStyle are **Wrapped**, **WrappedRequest**, **WrappedResponse**.

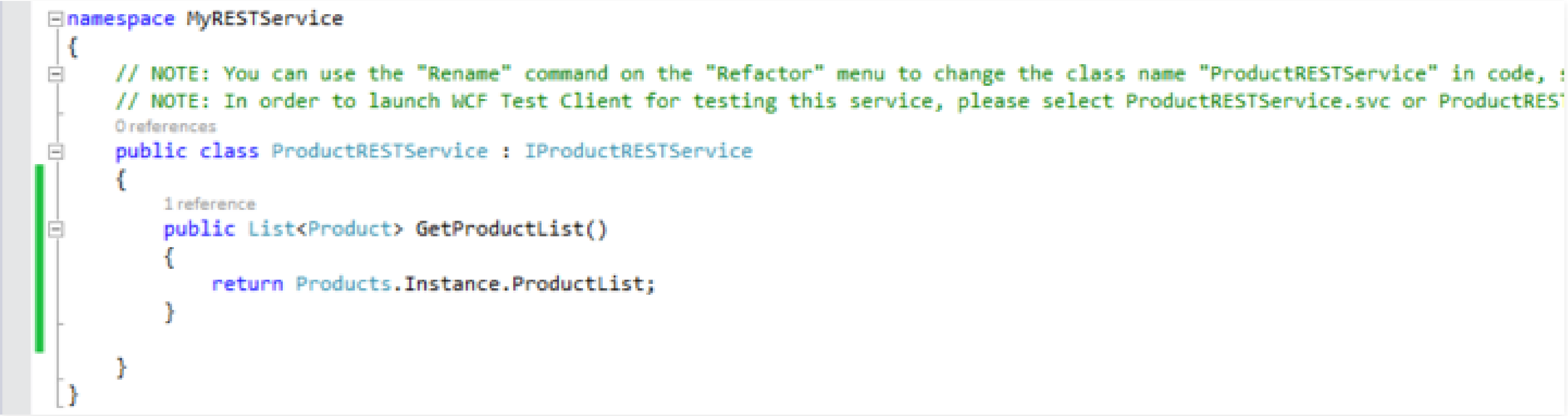
UriTemplate = “GetProductList/”, it has two parts, **URL path** and **query**.

Don’t forget to add **using System.ServiceModel.Web** at top.

7 REST-Based XML Web Service

4. Implementing RESTful Service

In this step we are going to implement the service. Only one method **GetProductList** is defined in the contract, so implementing service class will be as follows:



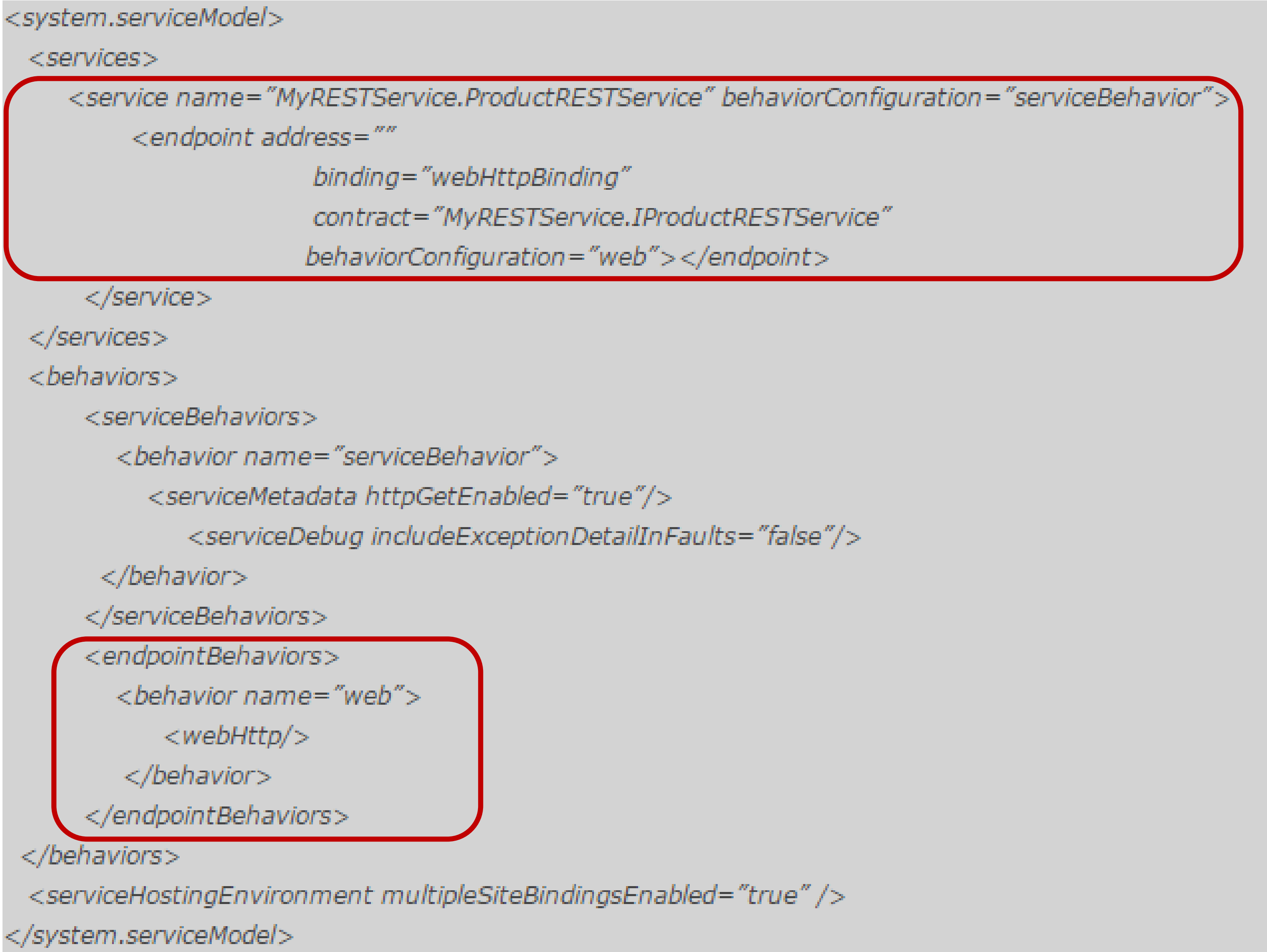
```
namespace MyRESTService
{
    // NOTE: You can use the "Rename" command on the "Refactor" menu to change the class name "ProductRESTService" in code, :
    // NOTE: In order to launch WCF Test Client for testing this service, please select ProductRESTService.svc or ProductREST
    0 references
    public class ProductRESTService : IProductRESTService
    {
        1 reference
        public List<Product> GetProductList()
        {
            return Products.Instance.ProductList;
        }

    }
}
```

7 REST-Based XML Web Service

5. Configure Service and Behavior

The last step is to configure the service and its behaviors using the configuration file **web.config**. Following is the complete **ServiceModel** configuration settings.

7 REST-Based XML Web Service



```
<system.serviceModel>
  <services>
      <service name=”MyRESTService.ProductRESTService” behaviorConfiguration=”serviceBehavior”>
            <endpoint address=””
                              binding=”webHttpBinding”
                              contract=”MyRESTService.IProductRESTService”
                             behaviorConfiguration=”web”></endpoint>
        </service>
  </services>
  <behaviors>
        <serviceBehaviors>
           <behavior name=”serviceBehavior”>
              <serviceMetadata httpGetEnabled=”true”/>
                  <serviceDebug includeExceptionDetailInFaults=”false”/>
         </behavior>
        </serviceBehaviors>
        <endpointBehaviors>
           <behavior name=”web”>
                <webHttp/>
             </behavior>
        </endpointBehaviors>
 </behaviors>
  <serviceHostingEnvironment multipleSiteBindingsEnabled=”true” />
</system.serviceModel>
```

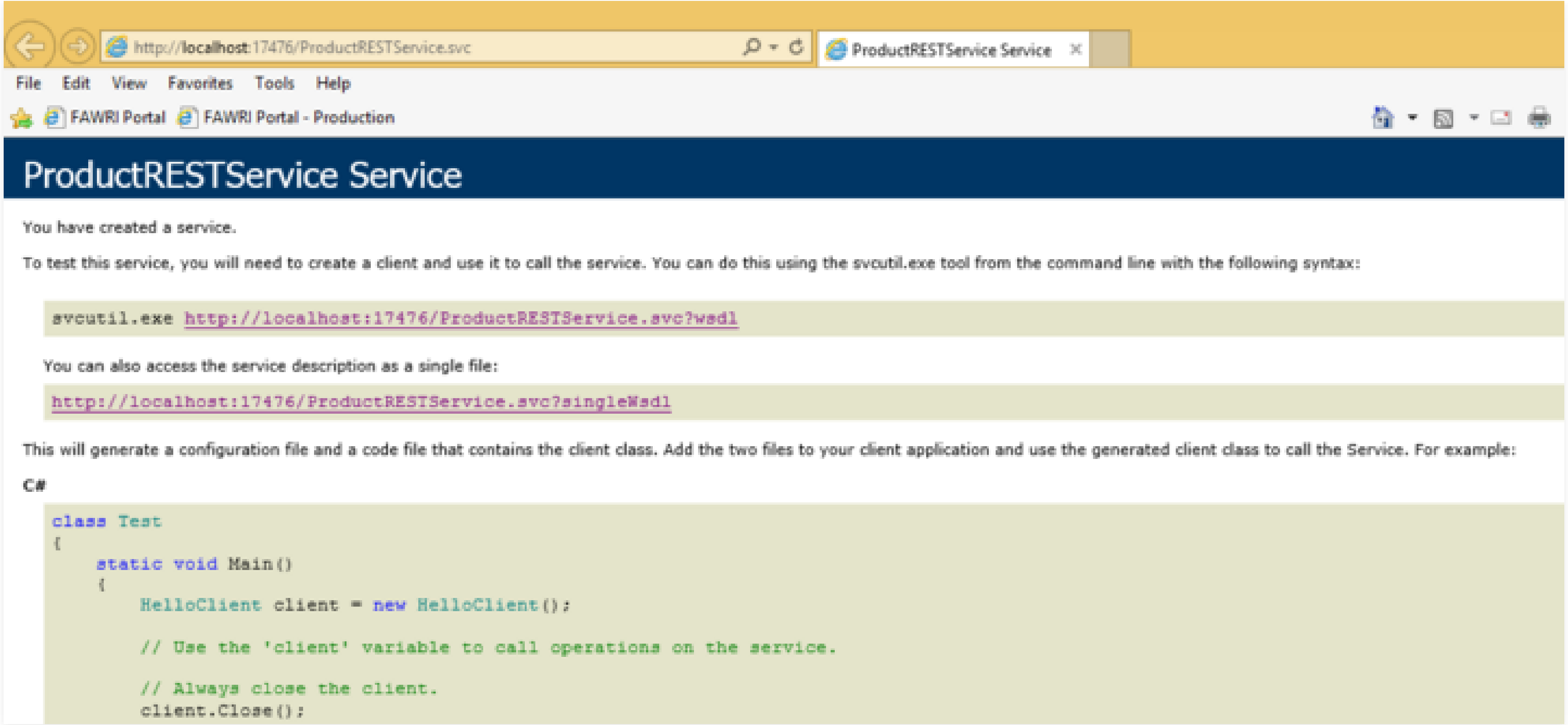
7 REST-Based XML Web Service

webHTTPBinding is the binding used for RESTful services.

Now, everything about creating RESTful service is done. You can easily run and test it.

Right click ***ProductRESTService.svc*** file and click “***View in Browser***“. You will see the following screen, that means service is fine.

7 REST-Based XML Web Service


http://localhost:17476/ProductRESTService.svc
ProductRESTService Service
File
Edit
View
Favorites
Tools
Help
FAWRI Portal
FAWRI Portal - Production

ProductRESTService Service

You have created a service.

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

```
svcutil.exe http://localhost:17476/ProductRESTService.svc?wsdl
```

You can also access the service description as a single file:

```
http://localhost:17476/ProductRESTService.svc?singleWsdl
```

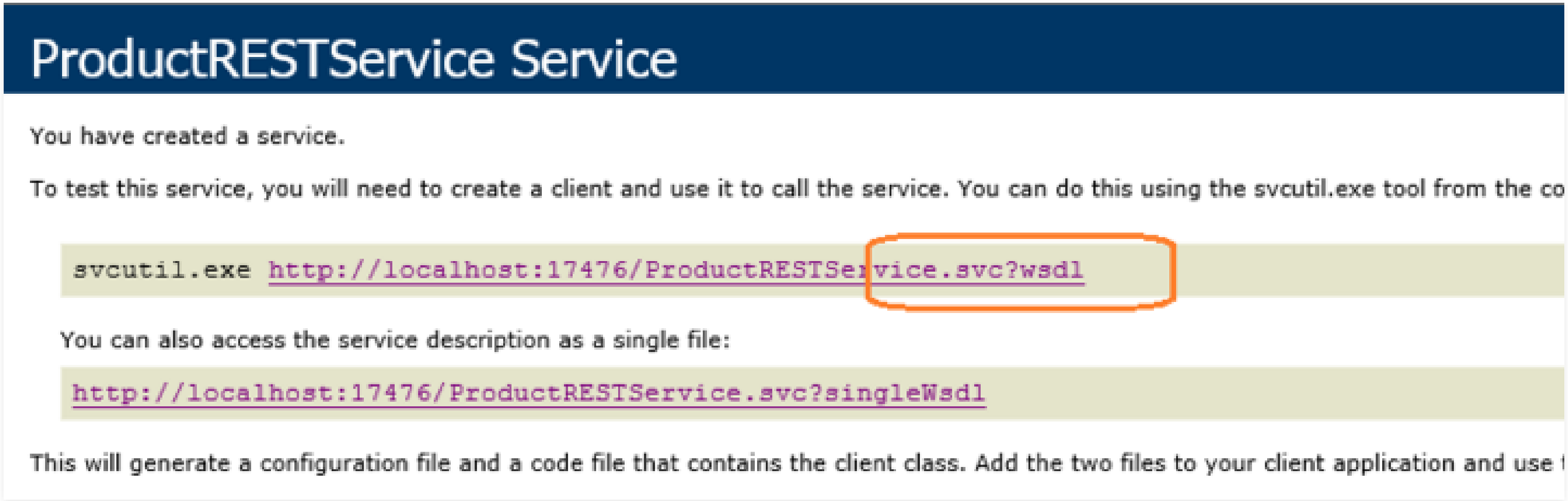
This will generate a configuration file and a code file that contains the client class. Add the two files to your client application and use the generated client class to call the Service. For example:

C#

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

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

        // Always close the client.
        client.Close();
```

7 REST-Based XML Web Service

> *An important point to consider here is that in **Service Behavior Configuration**, we have setted **httpGetEnabled="true"** for **serviceMetadata** that's why we are getting above service screen with wsdl option.*



ProductRESTService Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.exe tool from the co

```
svcutil.exe http://localhost:17476/ProductRESTService.svc?wsdl
```

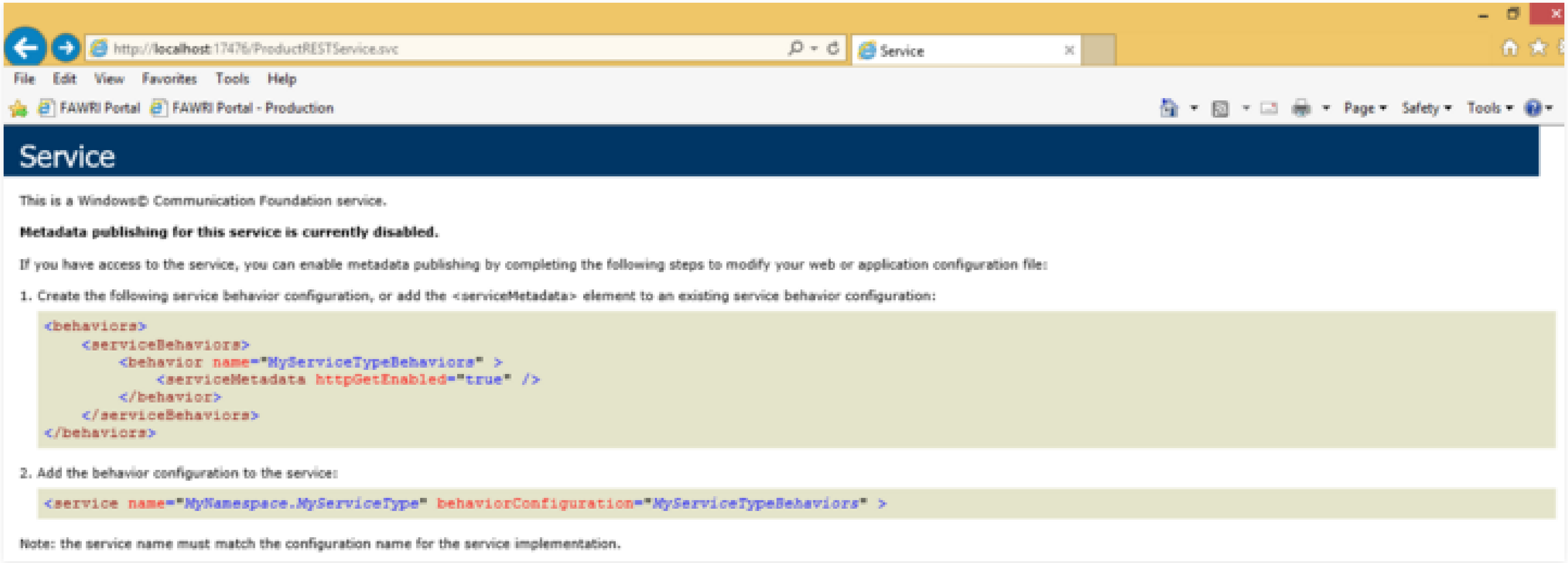
You can also access the service description as a single file:

```
http://localhost:17476/ProductRESTService.svc?singleWsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files to your client application and use t

7 REST-Based XML Web Service

This is what we do normally for SOAP-based services. REST-based services only uses HTTP verbs on a resource, so we can disable WSDL in this case by simply setting ***httpGetEnabled=”false”***. Now if we run the service again, we will get the following screen.


http://localhost:17476/ProductRESTService.svc
Service
File Edit View Favorites Tools Help
FAWRI Portal
FAWRI Portal - Production
Page ▾ Safety ▾ Tools ▾

Service

This is a Windows© Communication Foundation service.

Metadata publishing for this service is currently disabled.

If you have access to the service, you can enable metadata publishing by completing the following steps to modify your web or application configuration file:

1. Create the following service behavior configuration, or add the <serviceMetadata> element to an existing service behavior configuration:

```
<behaviors>
    <serviceBehaviors>
        <behavior name="MyServiceTypeBehaviors" >
            <serviceMetadata httpGetEnabled="true" />
        </behavior>
    </serviceBehaviors>
</behaviors>
```

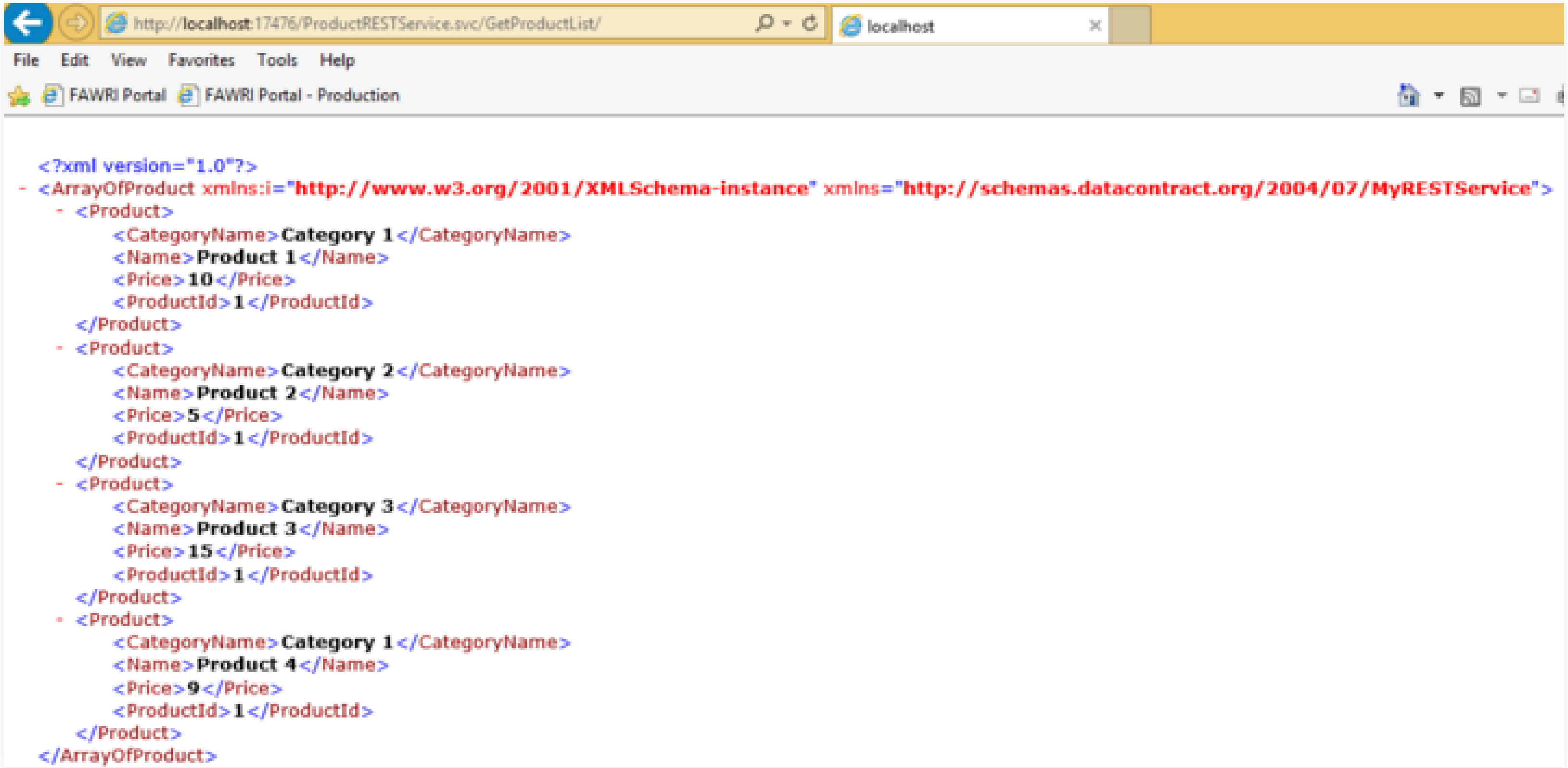
2. Add the behavior configuration to the service:

```
<service name="MyNamespace.MyServiceType" behaviorConfiguration="MyServiceTypeBehaviors" >
```

Note: the service name must match the configuration name for the service implementation.

7 REST-Based XML Web Service

Just modify the URL in browser and add “**GetProductList/**” to it. So, this is the UriTemplate we defined as service contract method.


http://localhost:17476/ProductRESTService.svc/GetProductList/
localhost
File Edit View Favorites Tools Help
FAWRI Portal
FAWRI Portal - Production

```
  <?xml version="1.0"?>
- <ArrayOfProduct xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/MyRESTService">
  - <Product>
        <CategoryName>Category 1</CategoryName>
        <Name>Product 1</Name>
        <Price>10</Price>
        <ProductId>1</ProductId>
    </Product>
  - <Product>
        <CategoryName>Category 2</CategoryName>
        <Name>Product 2</Name>
        <Price>5</Price>
        <ProductId>1</ProductId>
    </Product>
  - <Product>
        <CategoryName>Category 3</CategoryName>
        <Name>Product 3</Name>
        <Price>15</Price>
        <ProductId>1</ProductId>
    </Product>
  - <Product>
        <CategoryName>Category 1</CategoryName>
        <Name>Product 4</Name>
        <Price>9</Price>
        <ProductId>1</ProductId>
    </Product>
  </ArrayOfProduct>
```

7.1 Creating a REST-Based XML Web Service

- Create a new **WCF Service** project.
- `IWelcomeRESTXMLService` interface (Fig. 15b.11) is a modified version of the `IWelcomeSOAPXMLService` interface.

IWelcomeRESTXMLService.cs

```
// Fig. 23.11: IWelcomeRESTXMLService.cs
// WCF web-service interface. A class that implements this interface
// returns a welcome message through REST architecture and XML data
// format.
using System.ServiceModel;
using System.ServiceModel.Web;

[ServiceContract]
public interface IWelcomeRESTXMLService
{
   // returns a welcome message
   [OperationContract]
   [WebGet( UriTemplate = "/welcome/{yourName}" )]
   string Welcome( string yourName );
} // end interface IWelcomeRESTXMLService
```

The `WebGet` attribute maps a method to a unique URL.

Fig. 15b.11 | WCF web-service interface. A class that implements this interface returns a welcome message through REST architecture and XML data format.

7 Publishing and Consuming REST-Based XML Web Services

The `WebGet` attribute maps a method to a unique URL.

`WebGet`’s `UriTemplate` property specifies the URI format that is used to invoke the method.

http://www.topwcftutorials.net/2013/09/simple-steps-for-restful-service.html

7 Publishing and Consuming REST-Based XML Web Services

The `WebGet` attribute maps a method to a unique URL.

`WebGet`'s `UriTemplate` property specifies the URI format that is used to invoke the method.

http://www.topwcftutorials.net/2013/09/simple-steps-for-restful-service.html

Outline

- `WelcomeRESTXMLService` (Fig. 15b.12) is the class that implements the `IWelcomeRESTXMLService` interface; it is similar to the `WelcomeSOAPXMLService` class (Fig. 15b.2).

`WelcomeRESTXML`
`Service.cs`

```
// Fig. 23.12: WelcomeRESTXMLService.cs
// WCF web service that returns a welcome message using REST architecture
// and XML data format.
public class WelcomeRESTXMLService : IWelcomeRESTXMLService
{
   // returns a welcome message
   public string Welcome( string yourName )
   {
      return string.Format( "Welcome to WCF Web Services"
         + " with REST and XML, {0}!", yourName );
   } // end method Welcome
} // end class WelcomeRESTXMLService
```

Fig. 15b.12 | WCF web service that returns a welcome message using REST architecture and XML data format.

- Figure 15b.13 shows part of the default `Web.config` file modified to use REST architecture.

(1 of 2)

```
<system.serviceModel>
   <services>
      <service name="WelcomeRESTXMLService"
         behaviorConfiguration="ServiceBehavior">
         <!-- Service Endpoints -->
         <endpoint address="" binding="webHttpBinding"
            contract="IWelcomeRESTXMLService"
            behaviorConfiguration="RESTBehavior">
            <identity>
               <dns value="localhost"/>
            </identity>
         </endpoint>
         <endpoint address="mex" binding="mexHttpBinding"
            contract="IMetadataExchange"/>
      </service>
   </services>
```

`webHttpBinding` is used to respond to REST-based requests.

The `behaviorConfiguration` defines the endpoint’s behavior.

Fig. 15b.13 | `WelcomeRESTXMLService Web.config` file. (Part 1 of 2.)

(2 of 2)

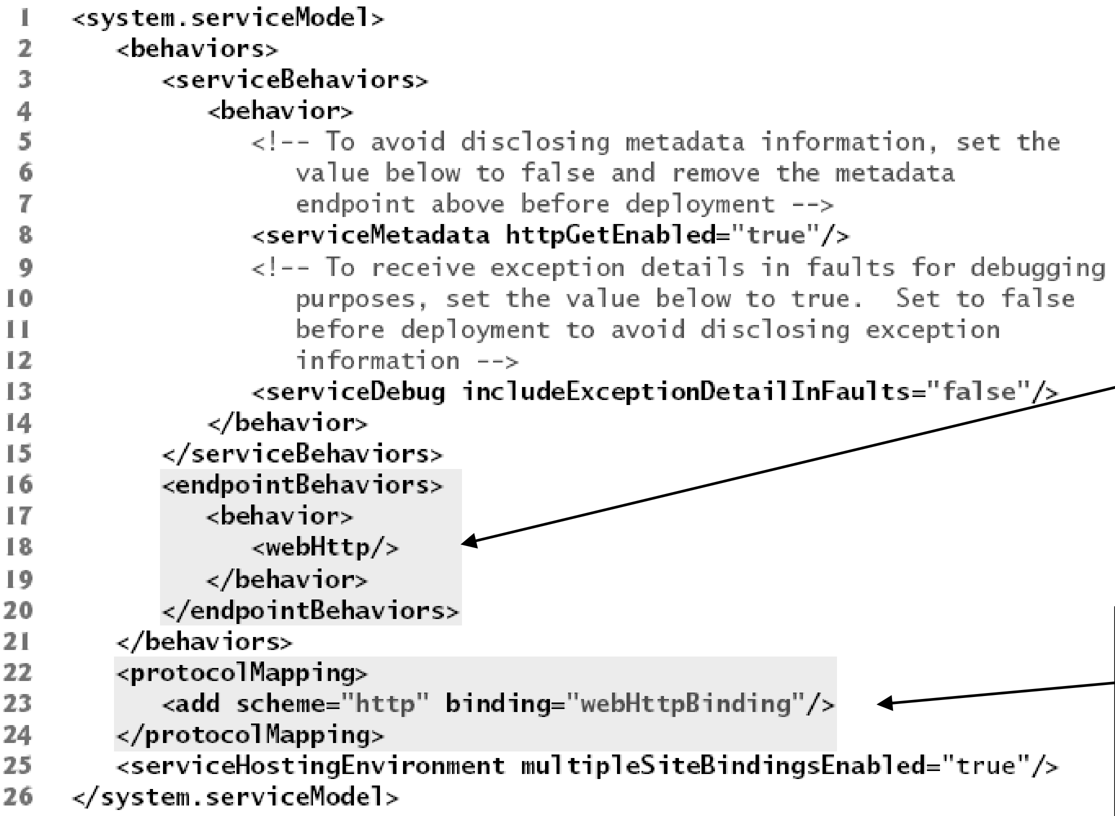
```
   <behaviors>
      <serviceBehaviors>
         <behavior name="ServiceBehavior">
            <serviceMetadata httpGetEnabled="true"/>
            <serviceDebug includeExceptionDetailInFaults="false"/>
         </behavior>
      </serviceBehaviors>
      <endpointBehaviors>
         <behavior name="RESTBehavior">
            <webHttp />
         </behavior>
      </endpointBehaviors>
   </behaviors>
</system.serviceModel>
```

Defining `RESTBehavior`, and specifying that clients communicate using HTTP.

Fig. 15b.13 | `WelcomeRESTXMLService` `Web.config` file. (Part 2 of 2.)

- Figure 15b.13 shows part of the default `web.config` file modified to use REST architecture.

(1 of 2)



```
<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>
   <serviceHostingEnvironment multipleSiteBindingsEnabled="true"/>
</system.serviceModel>
```

A must to Add:

The nested `webHttp` element specifies that clients communicate with this service using the standard **HTTP** request/response mechanism The `behaviorConfiguration` defines the endpoint’s behavior.

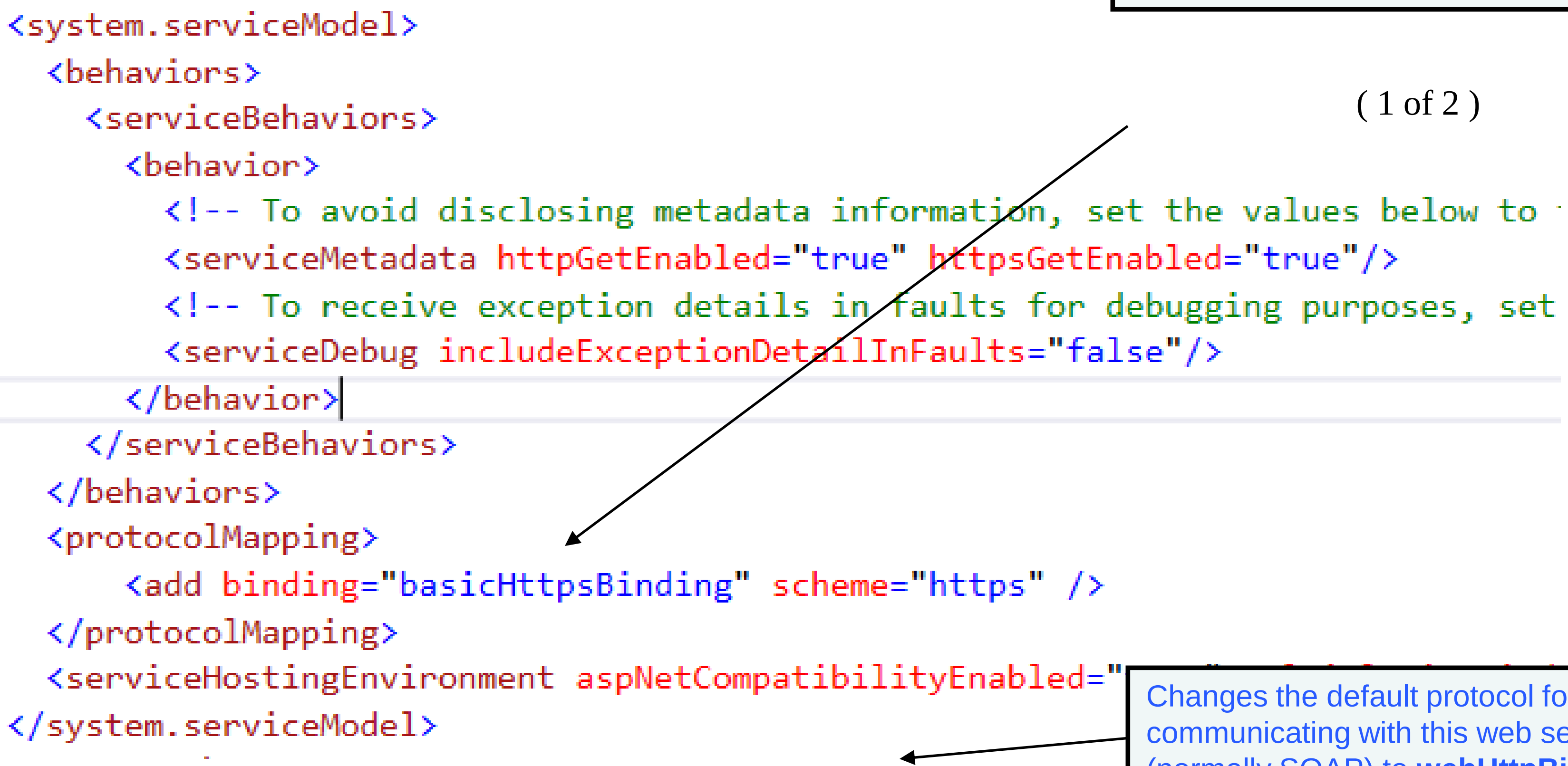
Changes the default protocol for communicating with this web service (normally SOAP) to **webHttpBinding, which is used for RESTbased** HTTP requests

Fig. 15b.13 | `WelcomeRESTXMLService` `Web.config` file (VS 2010).)

- Compare with SOAP `Web.config`

Uses basicHttpsBinding instead of webHttpBinding

(1 of 2)

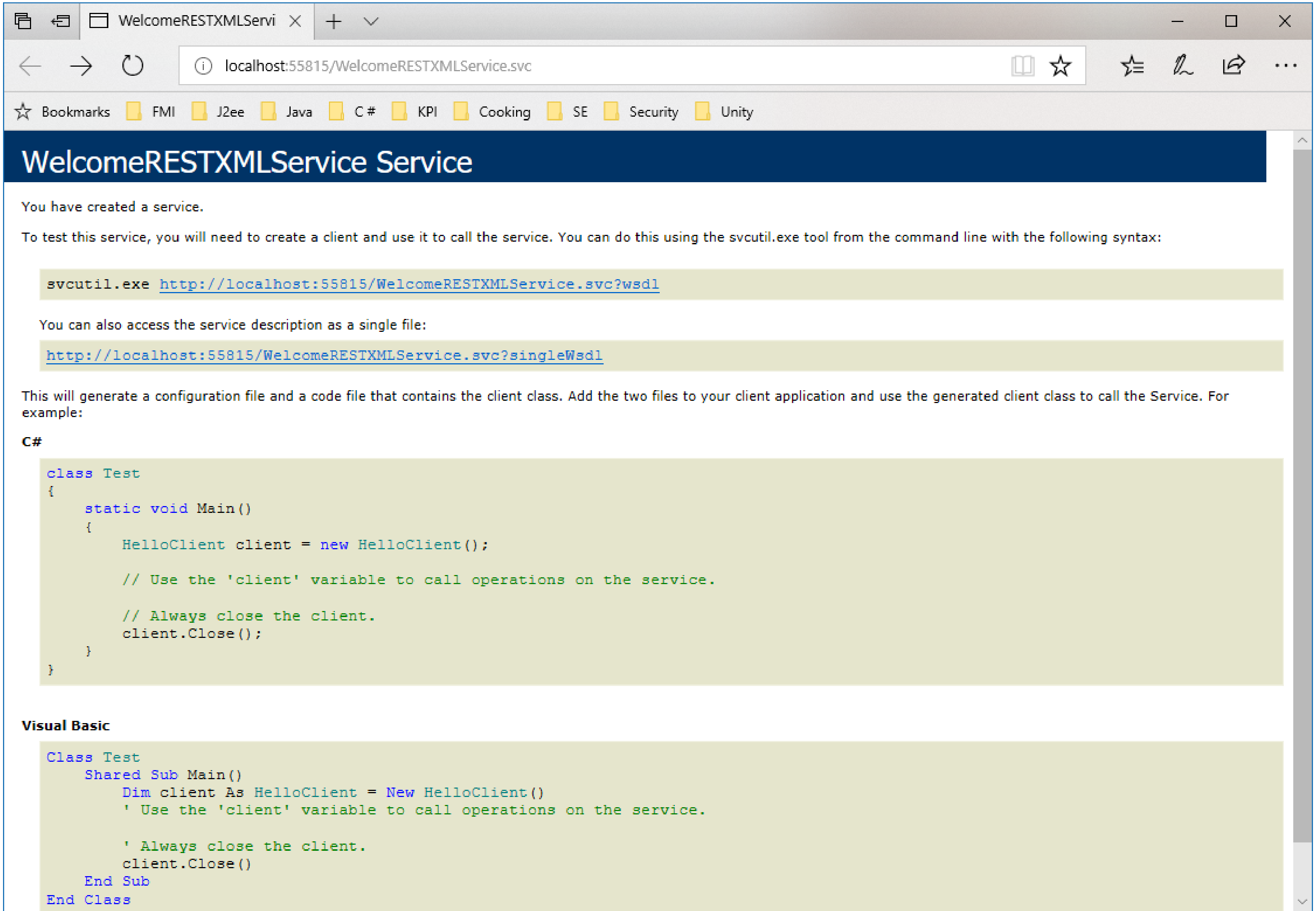


```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior>
        <!-- To avoid disclosing metadata information, set the values below to
        <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"/>
        <!-- To receive exception details in faults for debugging purposes, set
        <serviceDebug includeExceptionDetailInFaults="false"/>
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <protocolMapping>
      <add binding="basicHttpsBinding" scheme="https" />
  </protocolMapping>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="
</system.serviceModel>
```

Changes the default protocol for communicating with this web service (normally SOAP) to **webHttpBinding, which is used for RESTbased** HTTP requests

Fig. 15b.13 | `WelcomeRESTXMLService Web.config` file (VS 2010).)

View in Browser WelcomeRESTXMLService.svc



WelcomeRESTXMLService Service

You have created a service.

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

```
svcutil.exe http://localhost:55815/WelcomeRESTXMLService.svc?wsdl
```

You can also access the service description as a single file:

```
http://localhost:55815/WelcomeRESTXMLService.svc?singleWsdl
```

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

C#

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

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

        // Always close the client.
        client.Close();
    }
}
```

Visual Basic

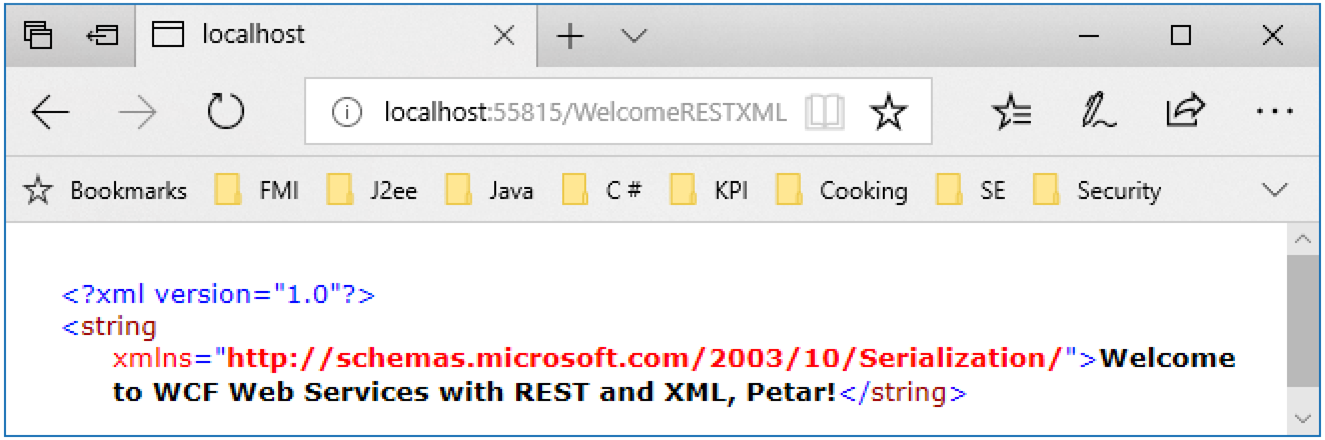
```
Class Test
    Shared Sub Main()
        Dim client As HelloClient = New HelloClient()
        ' Use the 'client' variable to call operations on the service.

        ' Always close the client.
        client.Close()
    End Sub
End Class
```

7 Publishing and Consuming REST-Based XML Web Services (Cont.)

Figure 15b.14 tests the `WelcomeRESTXMLService`’s `Welcome` method in a web browser by following the URI template.

`http://localhost:55815/WelcomeRESTXMLService/Service.svc/welcome/Bruce.`

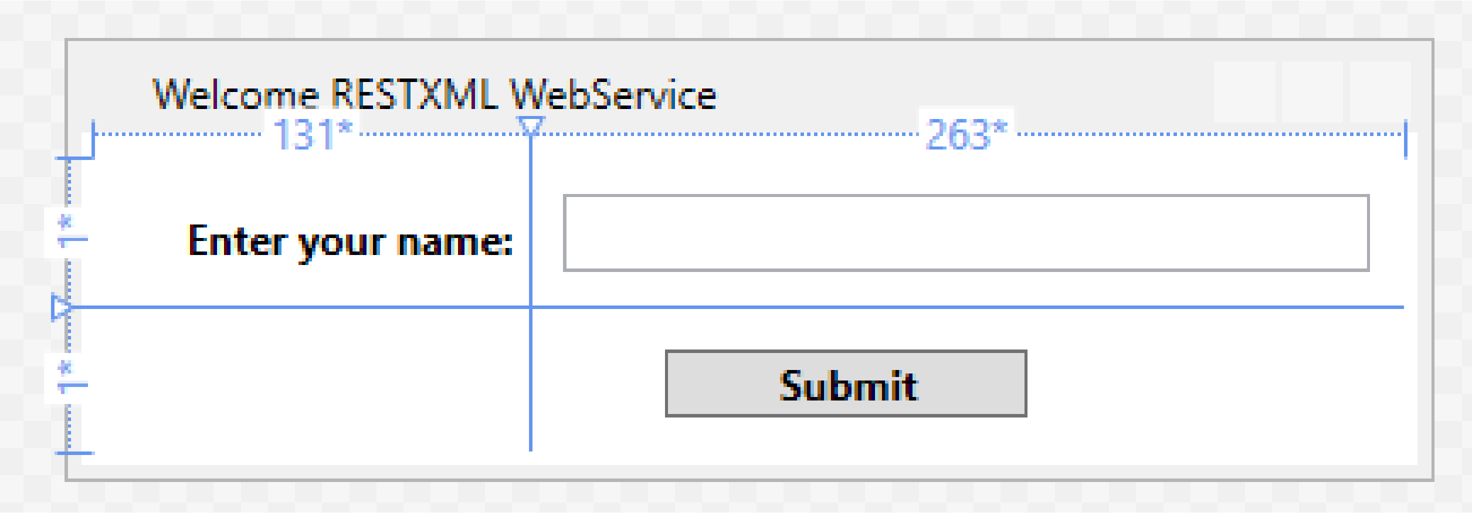
The browser displays the XML data response from `WelcomeRESTXMLService`.



```
<?xml version="1.0"?>
<string
    xmlns="http://schemas.microsoft.com/2003/10/Serialization/">Welcome
    to WCF Web Services with REST and XML, Petar!</string>
```

Fig. 15b.14 | Response from `WelcomeRESTXMLService` in XML data format.

7.1 Create a WPF client


Welcome RESTXML WebService

Enter your name:

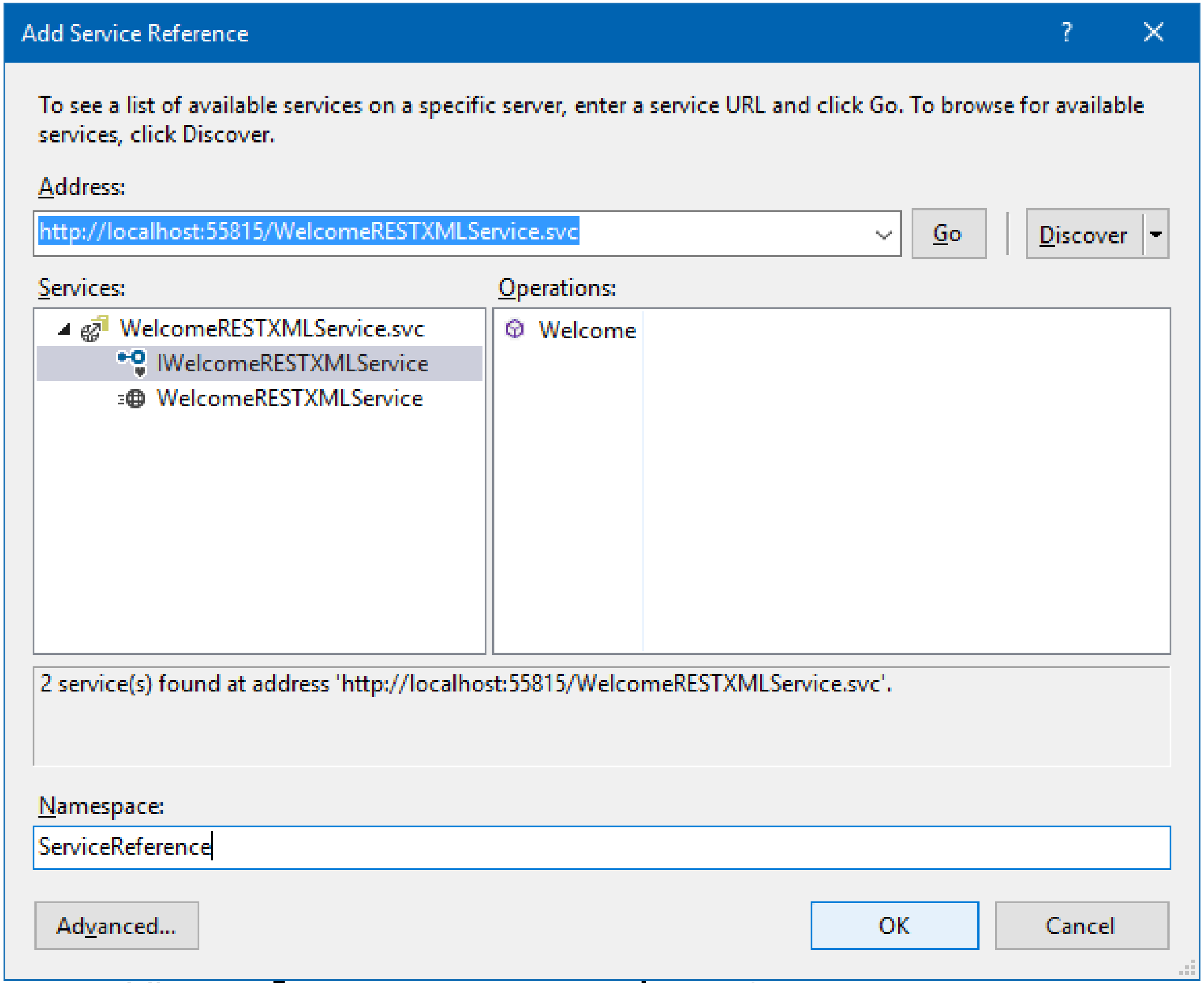
Fig. 15b.14 | Response from `WelcomeRESTXMLService` in XML data format.

```
<Window x:Class="WelcomeRESTXMLClientWPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WelcomeRESTXMLClientWPF“ mc:Ignorable="d"
        Title="Welcome RESTXML WebService" Height="129" Width="402">
    <Grid Height="86" VerticalAlignment="Bottom" Margin="4">
        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>   <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="131*"/> <ColumnDefinition Width="263*"/>
        </Grid.ColumnDefinitions>
        <Button x:Name="BtnSubmit" Content="Submit" HorizontalAlignment="Center“
          Margin="58,13,0,0" Grid.Row="1" VerticalAlignment="Top" Width="107“
          FontWeight="Bold" Height="20" Grid.ColumnSpan="2" Click="BtnSubmit_Click" />
        <Label x:Name="LblEnterName" Content="Enter your name:" HorizontalAlignment="Left“
         Margin="22,10,0,0" VerticalAlignment="Top" Width="125" FontWeight="Bold“
         Grid.ColumnSpan="2" Height="26"/>
        <TextBox x:Name="TxtYourName" Grid.Column="1" HorizontalAlignment="Left“
          Height="23" Margin="10,10,0,0" VerticalAlignment="Top" Width="238“
          FontWeight="Bold"/>
    </Grid>
</Window>
```

Add a web service reference to the client project using

`http://localhost:55815/WelcomeRESTXLService/Service.svc`

to discover the REST web service



Add Service Reference

To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

Address: http://localhost:55815/WelcomeRESTXMLService.svc

Services:

- WelcomeRESTXMLService.svc
  - IWelcomeRESTXMLService
  - WelcomeRESTXMLService

Operations:

- Welcome

2 service(s) found at address 'http://localhost:55815/WelcomeRESTXMLService.svc'.

Namespace: ServiceReference

Fig. 15b.14 | Adding `WelcomeRESTXMLService` reference

WelcomeRESTXMLClientWPF (Fig. 15b.15) invokes the web service and receive its response.

WelcomeRESTXML Form.cs

(1 of 4)

```
// MainWindow.xaml.cs
// Client that consumes the WelcomeRESTXMLService.
using System;
using System.Net.Http;
using System.Windows;
using System.Xml.Linq;

namespace WelcomeRESTXMLClient
{
   public partial class MainWindow : Window
   {
      // object to invoke the WelcomeRESTXMLService
      private HttpClient client = new HttpClient();

      private XNamespace xmlNamespace =
         XNamespace.Get(
         "http://schemas.microsoft.com/2003/10/Serialization/" );
```

Using the System.Net.Http namespace's HttpClient

Fig. 15b.15 | Client that consumes the WelcomeRESTXMLService. (Part 1 of 4.)

```

      public MainWindow()
      {
         InitializeComponent();
      } // end constructor

      // get user input and pass it to the web service
      private async void BtnSubmit_Click( object sender, RoutedEventArgs e )
      {
       // send request to WelcomeRESTXMLService
      string result = await client.GetStringAsync(new Uri(
      "http://localhost:55815/WelcomeRESTXMLService.svc/welcome/" +
      TxtYourName.Text));
      // parse the returned XML
       XDocument xmlResponse = XDocument.Parse(result);
        // get the <string> element's value
         MessageBox.Show(xmlResponse.Element(
         xmlNamespace + "string").Value, "Welcome");
      } // end method BtnSubmit_Click
   } // end of class
} // end of namespace
```

WelcomeRESTXMLForm.cs

(2 of 4)

The client's GetStringAsync method invokes the web service asynchronously.

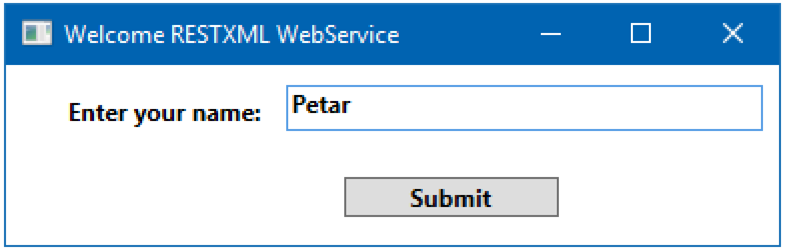
Parse the string element in the response.

Fig. 15b.15 | Client that consumes the WelcomeRESTXMLService. (Part 2 of 4.)

`WelcomeRESTXMLClientWPF`

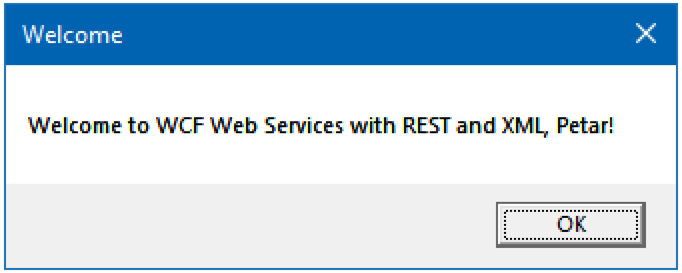
(4 of 4)

a) User inputs name.



Enter your name: Petar

b) Message sent from WelcomeRESTXMLService.



Welcome to WCF Web Services with REST and XML, Petar!

Fig. 15b.15 | Client that consumes the `WelcomeRESTXMLService`. (Part 4 of 4.)

7 Publishing and Consuming REST-Based XML Web Services (Cont.)

The `client`’s `GetStringAsync` method invokes the web service asynchronously.

When the call to the web service completes, the `HttpClient` object returns the XML string with the response.

`XDocument` uses method `Parse` to create a XML document from the response, from which we retrieve the `Value` of the element named “`string`”.

Outline

- In Fig. 15b.16, we modify the `Welcome-RESTXMLService` to return data in JSON format.

IWelcomeRESTJSONService.cs

(1 of 2)

```
// Fig. 23.16: IWelcomeRESTJSONService.cs
// WCF web-service interface that returns a welcome message through REST
// architecture and JSON format.
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;

[ServiceContract]
public interface IWelcomeRESTJSONService
{
   // returns a welcome message
   [OperationContract]
   [WebGet( ResponseFormat = WebMessageFormat.Json,
      UriTemplate = "/welcome/{yourName}" )]
   TextMessage Welcome( string yourName );
} // end interface IWelcomeRESTJSONService
```

Set the `ResponseFormat` property to `WebMessageFormat.Json`.

Fig. 15b.16 | WCF web-service interface that returns a welcome message through REST architecture and JSON format. (Part 1 of 2.)

Outline

IWelcomeRESTJSONService.cs

(2 of 2)

```

// class to encapsulate a string to send in JSON format
[DataContract]
public class TextMessage
{
   // automatic property message
   [DataMember]
   public string Message {get; set; }
} // end class TextMessage
```

The `DataContract` attribute exposes the `TextMessage` class for serialization.

The `DataMember` attribute exposes a property of this class for serialization.

Fig. 15b.16 | WCF web-service interface that returns a welcome message through REST architecture and JSON format. (Part 2 of 2.)

8 Publishing and Consuming REST-Based JSON Web Services (Cont.)

For JSON serialization to work properly, the objects being converted to JSON must have `public` properties.

`string`s do not have `public` properties, so a serializable `TextMessage` class was used in this example.

- Figure 15b.17 shows the implementation of the interface of Fig. 15b.16.

WelcomeRESTJSON Service.cs

```
// Fig. 23.17: WelcomeRESTJSONService.cs
// WCF web service that returns a welcome message through REST
// architecture and JSON format.
public class WelcomeRESTJSONService : IWelcomeRESTJSONService
{
   // returns a welcome message
   public TextMessage Welcome( string yourName )
   {
      // add welcome message to field of TextMessage object
      TextMessage message = new TextMessage();
      message.Message = string.Format( "Welcome to WCF Web Services" +
         " with REST and JSON, {0}!", yourName );
      return message;
   } // end method Welcome
} // end class WelcomeRESTJSONService
```

The `Welcome` method returns a `TextMessage` object, automatically serialized in JSON format.

Fig. 15b.17 | WCF web service that returns a welcome message through REST architecture and JSON format.

8 Publishing and Consuming REST-Based JSON Web Services (Cont.)

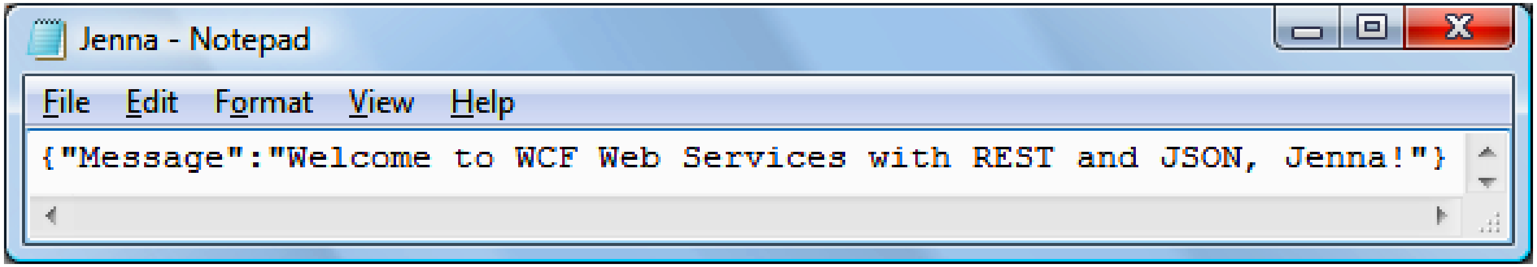
Test the web service by accessing the `Service.svc` file:

`http://localhost:56429/WelcomeRESTJSON Service/Service.svc`

Append the URI template (`welcome`/*yourName*) to the address.

http://localhost:56429/WelcomeRESTJSONService.svc/welcome/Jenna

The service response is a JSON object (Fig. 15b.18).



```
{"Message":"Welcome to WCF Web Services with REST and JSON, Jenna!"}
```

Fig. 15b.18 | Response from `WelcomeRESTJSONService` in JSON data format.

`WelcomeRESTJSONForm.cs`

(1 of 5)

Custom types that are sent to or from a REST web service are converted using **XML** or **JSON serialization**.

In Fig. 15b.19, we consume the JSON web service.

Right click the project name, select

- Add Service Reference to add the REST web service. Discover the web service by http:// localhost:56429/WelcomeRESTJSONService.svc
- Add Reference also to `System.Net.Http` and `System.Runtime.Serialization.Json`

Outline

WelcomeRESTJSONClientWPF

```
// MainWindow.xaml.cs
// Client that consumes WelcomeRESTJSONService.
using System;
using System.IO;
using System.Net.Http;
using System.Runtime.Serialization.Json;
using System.Text;
using System.Windows.Windows;

namespace WelcomeRESTJSONClientWPF
{
   public partial class MainWindow : Window
   {
      // object to invoke the WelcomeRESTJSONService
      private HttpClient client = new HttpClient();

```

Fig. 15b.19 | Client that consumes `WelcomeRESTJSONService`. (Part 1 of 3.)

WelcomeRESTJSONClientWPF

```
public MainWindow()
{
   InitializeComponent();
}// end constructor
// get user input, pass it to web service, and process response
private void BtnSubmit_Click( object sender, RoutedEventArgs e )
{  // send request to WelcomeRESTXMLService
   string result = await client.GetStringAsync(new Uri(
  "http://localhost:56429/WelcomeRESTJSONService.svc/welcome/" +
   TxtYourName.Text));
  //  deserialize response into a TextMessage object
  DataContractJsonSerializer JSONSerializer =
          new DataContractJsonSerializer(typeof(TextMessage));
  TextMessage message =
          new MemoryStream(Encoding.Unicode.GetBytes(result)));
  // display Message text
  MessageBox.Show(message.Message, "Welcome");
} // end method submitButton_Click

```

Creating an object for performing JSON serialization on `TextMessage` objects.

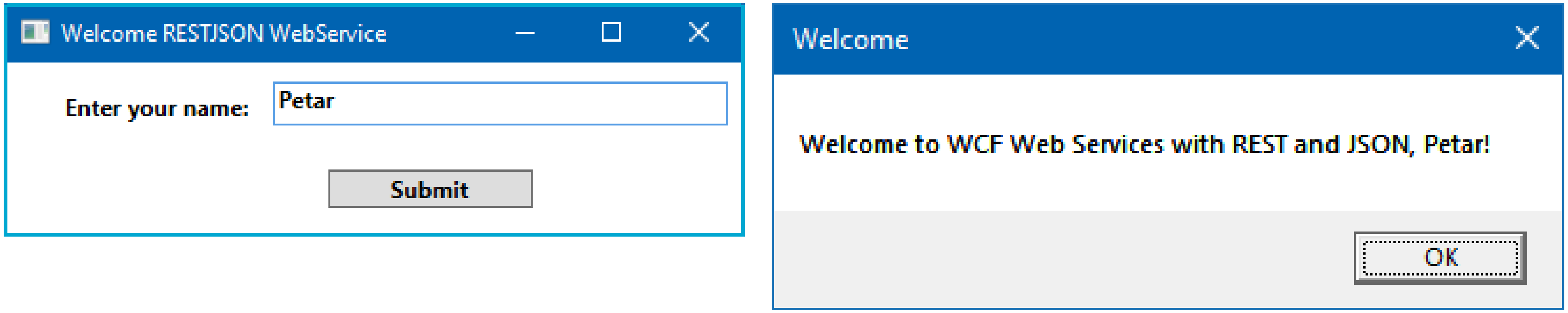
Using the `GetBytes` method to convert the JSON response to a stream.

Fig. 15b.19 | Client that consumes `WelcomeRESTJSONService`. (Part 2 of 3.)

WelcomeRESTJSONClientWPF

```
   // TextMessage class representing a JSON object
   [Serializable]
   public class TextMessage
   {
      public string Message;
   } // end class TextMessage
} // end namespace WelcomeRESTJSONClient
```

The `TextMessage` class maps fields for JSON serialization.



a) User inputs name.

Welcome RESTJSON WebService

Enter your name: Petar

b) Message sent from **WelcomeRESTJSONService**.

Welcome

Welcome to WCF Web Services with REST and JSON, Petar!

Fig. 15b.19 | Client that consumes `WelcomeRESTJSONService`. (Part 3 of 3.)

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

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

A session variable allows web-service methods to return personalized, localized results.

Outline

- You will now create a WCF web service that follows a simple subset of casino blackjack rules.
- The web service's interface is defined in Fig. 15b.20.

IBlackjackService.cs

```
// Fig. 23.20: IBlackjackService.cs
// Blackjack game WCF web-service interface.
using System.ServiceModel;

[ServiceContract( SessionMode = SessionMode.Required )]
public interface IBlackjackService
{
   // deals a card that has not been dealt
   [OperationContract]
   string DealCard();

   // creates and shuffles the deck
   [OperationContract]
   void Shuffle();

   // calculates value of a hand
   [OperationContract]
   int GetHandValue( string dealt );
} // end interface IBlackjackService
```

The service requires sessions to execute correctly.

Fig. 15b.20 | Blackjack game WCF web-service interface.

- The web-service class (Fig. 15b.21) provides methods to deal a card, shuffle the deck and determine the point value of a hand.

BlackjackService.cs

(1 of 5)

```
// Fig. 23.21: BlackjackService.cs
// Blackjack game WCF web service.
using System;
using System.Collections.Generic;
using System.ServiceModel;

[ServiceBehavior( InstanceContextMode = InstanceContextMode.PerSession )]
public class BlackjackService : IBlackjackService
{
   // create persistent session deck-of-cards object
   List< string > deck = new List< string >();

   // deals card that has not yet been dealt
   public string DealCard()
   {
      string card = deck[ 0 ]; // get first card
      deck.RemoveAt( 0 ); // remove card from deck
      return card;
   } // end method DealCard

```

Setting the ServiceBehavior's InstanceContextMode property to PerSession creates a new instance of the class for each session.

DealCard manipulates the current user's deck by returning the top card's value.

Fig. 15b.21 | Blackjack game WCF web service. (Part 1 of 5)

```
   // creates and shuffles a deck of cards
   public void Shuffle()
   {
      Random randomObject = new Random(); // generates random numbers

      deck.Clear(); // clears deck for new game

      // generate all possible cards
      for ( int face = 1; face <= 13; face++ ) // loop through faces
         for ( int suit = 0; suit <= 3; suit++ ) // loop through suits
            deck.Add( face + " " + suit ); // add card (string) to deck

      // shuffles deck by swapping each card with another card randomly
      for ( int i = 0; i < deck.Count; i++ )
      {
         // get random index
         int newIndex = randomObject.Next( deck.Count - 1 );

         // save current card in temporary variable
         string temporary = deck[ i ];
         deck[ i ] = deck[ newIndex ]; // copy randomly selected card
```

`BlackjackService.cs`

(2 of 5)

`Shuffle` fills the `List` object with `string`s representing a deck of cards.

Fig. 15b.21 | Blackjack game WCF web service. (Part 2 of 5)

```

         // copy current card back into deck
         deck[ newIndex ] = temporary;
      } // end for
   } // end method Shuffle

   // computes value of hand
   public int GetHandValue( string dealt )
   {
      // split string containing all cards
      string[] cards = dealt.Split( '\t' ); // get array of cards
      int total = 0; // total value of cards in hand
      int face; // face of the current card
      int aceCount = 0; // number of aces in hand

      // loop through the cards in the hand
      foreach ( var drawn in cards )
      {
         // get face of card
         face = Convert.ToInt32(
            drawn.Substring( 0, drawn.IndexOf( ' ' ) ) );
```

`BlackjackService.cs`

(3 of 5)

`Shuffle` fills the `List` object with `string`s representing a deck of cards.

Tokenizing the full hand of cards into an array of cards.

Counting the value of each card.

Fig. 15b.21 | Blackjack game WCF web service. (Part 3 of 5)

BlackjackService.cs

(4 of 5)

```

            switch ( face )
            {
               case 1: // if ace, increment aceCount
                  ++aceCount;
                  break;
               case 11: // if jack add 10
               case 12: // if queen add 10
               case 13: // if king add 10
                  total += 10;
                  break;
               default: // otherwise, add value of face
                  total += face;
                  break;
            } // end switch
         } // end foreach
```

Counting the number of aces.

Counting the value of each card.

Fig. 15b.21 | Blackjack game WCF web service. (Part 4 of 5)

```

      // if there are any aces, calculate optimum total
      if ( aceCount > 0 )
      {
         // if it is possible to count one ace as 11, and the rest
         // as 1 each, do so; otherwise, count all aces as 1 each
         if ( total + 11 + aceCount - 1 <= 21 )
            total += 11 + aceCount - 1;
         else
            total += aceCount;
      } // end if

      return total;
   } // end method GetHandValue
} // end class BlackjackService
```

BlackjackService.cs

(5 of 5)

Processing the aces after all the other cards (one ace can be counted as 11).

Fig. 15b.21 | Blackjack game WCF web service. (Part 5 of 5)

9 Blackjack Web Service: Using Session Tracking in a SOAP-Based Web Service (Cont.)

Setting the `ServiceBehavior`'s `InstanceContextMode` property to `PerSession` creates a new instance of the class for each session.

Method `Split` uses a delimiter character to divide a `string` into an array of substrings.

Outline

- Now we use our blackjack web service in a Windows application (Fig. 15b.22).
- You must add a service reference to your project so it can access the web service.

`BlackjackForm.cs`

(1 of 17)

```
// Fig. 23.22: BlackjackForm.cs
// Blackjack game that uses the BlackjackService web service.
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Resources;

namespace BlackjackClient
{
   public partial class BlackjackForm : Form
   {
      // reference to web service
      private ServiceReference.BlackjackServiceClient dealer;

```

Declaring the client object representing the dealer.

Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 1 of 17.)

```
      // string representing the dealer's cards
      private string dealersCards;

      // string representing the player's cards
      private string playersCards;

      // list of PictureBoxes for card images
      private List< PictureBox > cardBoxes;
      private int currentPlayerCard; // player's current card number
      private int currentDealerCard; // dealer's current card number

      private ResourceManager pictureLibrary =
         BlackjackClient.Properties.Resources.ResourceManager;

      // enum representing the possible game outcomes
      public enum GameStatus
      {
         PUSH, // game ends in a tie
         LOSE, // player loses
         WIN, // player wins
         BLACKJACK // player has blackjack
      } // end enum GameStatus
```

`BlackjackForm.cs`

(2 of 17)

Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 2 of 17.)

`BlackjackForm.cs`

(3 of 17)

```
      public BlackjackForm()
      {
         InitializeComponent();
      } // end constructor

      // sets up the game
      private void Blackjack_Load( object sender, EventArgs e )
      {
         // instantiate object allowing communication with web service
         dealer = new ServiceReference.BlackjackServiceClient();

         // put PictureBoxes into cardBoxes List
         cardBoxes = new List<PictureBox>(); // create list
         cardBoxes.Add( pictureBox1 );
         cardBoxes.Add( pictureBox2 );
         cardBoxes.Add( pictureBox3 );
         cardBoxes.Add( pictureBox4 );
         cardBoxes.Add( pictureBox5 );
         cardBoxes.Add( pictureBox6 );
         cardBoxes.Add( pictureBox7 );
         cardBoxes.Add( pictureBox8 );
         cardBoxes.Add( pictureBox9 );
```

Creating the client object representing the dealer.

Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 3 of 17.)

```
      cardBoxes.Add( pictureBox10 );
      cardBoxes.Add( pictureBox11 );
      cardBoxes.Add( pictureBox12 );
      cardBoxes.Add( pictureBox13 );
      cardBoxes.Add( pictureBox14 );
      cardBoxes.Add( pictureBox15 );
      cardBoxes.Add( pictureBox16 );
      cardBoxes.Add( pictureBox17 );
      cardBoxes.Add( pictureBox18 );
      cardBoxes.Add( pictureBox19 );
      cardBoxes.Add( pictureBox20 );
      cardBoxes.Add( pictureBox21 );
      cardBoxes.Add( pictureBox22 );
   } // end method BlackjackForm_Load

   // deals cards to dealer while dealer's total is less than 17,
   // then computes value of each hand and determines winner
   private void DealerPlay()
   {
      // reveal dealer's second card
      string[] cards = dealersCards.Split( '\t' );
      DisplayCard( 1, cards[1] );
```

`BlackjackForm.cs`

(4 of 17)

Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 4 of 17.)

BlackjackForm.cs

(5 of 17)

```
         string nextCard;

         // while value of dealer's hand is below 17,
         // dealer must take cards
         while ( dealer.GetHandValue( dealersCards ) < 17 )
         {
            nextCard = dealer.DealCard(); // deal new card
            dealersCards += '\t' + nextCard; // add new card to hand

            // update GUI to show new card
            MessageBox.Show( "Dealer takes a card" );
            DisplayCard( currentDealerCard, nextCard );
            ++currentDealerCard;
         } // end while

         int dealersTotal = dealer.GetHandValue( dealersCards );
         int playersTotal = dealer.GetHandValue( playersCards );

```

Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 5 of 17.)

BlackjackForm.cs

(6 of 17)

```
         // if dealer busted, player wins
         if ( dealersTotal > 21 )
         {
            GameOver( GameStatus.WIN );
         } // end if
         else
         {
            // if dealer and player have not exceeded 21,
            // higher score wins; equal scores is a push.
            if ( dealersTotal > playersTotal ) // player loses game
               GameOver( GameStatus.LOSE );
            else if ( playersTotal > dealersTotal ) // player wins game
               GameOver( GameStatus.WIN );
            else // player and dealer tie
               GameOver( GameStatus.PUSH );
         } // end else
      } // end method DealerPlay

```

Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 6 of 17.)

BlackjackForm.cs

(7 of 17)

```
   // displays card represented by cardValue in specified PictureBox
   public void DisplayCard( int card, string cardValue )
   {
      // retrieve appropriate PictureBox
      PictureBox displayBox = cardBoxes[ card ];

      // if string representing card is empty,
      // set displayBox to display back of card
      if ( string.IsNullOrEmpty( cardValue ) )
      {
         displayBox.Image =
            ( Image ) pictureLibrary.GetObject( "cardback" );
         return;
      } // end if

      // retrieve face value of card from cardValue
      string face =
         cardValue.Substring( 0, cardValue.IndexOf( ' ' ) );

```

Fig. 15b.22 | Blackjack game that uses the BlackjackService web service. (Part 7 of 17.)

```
// retrieve the suit of the card from cardValue
string suit =
   cardValue.Substring( cardValue.IndexOf( ' ' ) + 1 );

char suitLetter; // suit letter used to form image-file name

// determine the suit letter of the card
switch ( Convert.ToInt32( suit ) )
{
   case 0: // clubs
      suitLetter = 'c';
      break;
   case 1: // diamonds
      suitLetter = 'd';
      break;
   case 2: // hearts
      suitLetter = 'h';
      break;
   default: // spades
      suitLetter = 's';
      break;
} // end switch
```

`BlackjackForm.cs`

(8 of 17)

Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 8 of 17.)

```

      // set displayBox to display appropriate image
      displayBox.Image = ( Image ) pictureLibrary.GetObject(
         "_" + face + suitLetter );
   } // end method DisplayCard

   // displays all player cards and shows
   // appropriate game status message
   public void GameOver( GameStatus winner )
   {
      string[] cards = dealersCards.Split( '\t' );

      // display all the dealer's cards
      for ( int i = 0; i < cards.Length; i++ )
         DisplayCard( i, cards[ i ] );

      // display appropriate status image
      if ( winner == GameStatus.PUSH ) // push
         statusPictureBox.Image =
            ( Image ) pictureLibrary.GetObject( "tie" );
      else if ( winner == GameStatus.LOSE ) // player loses
```

BlackjackForm.cs

(9 of 17)

Fig. 15b.22 | Blackjack game that uses the BlackjackService web service. (Part 9 of 17.)

```
         statusPictureBox.Image =
            ( Image ) pictureLibrary.GetObject( "lose" );
      else if ( winner == GameStatus.BLACKJACK )
         // player has blackjack
         statusPictureBox.Image =
            ( Image ) pictureLibrary.GetObject( "blackjack" );
      else // player wins
         statusPictureBox.Image =
            ( Image ) pictureLibrary.GetObject( "win" );

      // display final totals for dealer and player
      dealerTotalLabel.Text =
         "Dealer: " + dealer.GetHandValue( dealersCards );
      playerTotalLabel.Text =
         "Player: " + dealer.GetHandValue( playersCards );

      // reset controls for new game
      stayButton.Enabled = false;
      hitButton.Enabled = false;
      dealButton.Enabled = true;
   } // end method GameOver

```

Displaying the final point totals of both the dealer and the player.

`BlackjackForm.cs`

(10 of 17)

Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 10 of 17.)

```
// deal two cards each to dealer and player
private void dealButton_Click( object sender, EventArgs e )
{
   string card; // stores a card temporarily until added to a hand

   // clear card images
   foreach ( PictureBox cardImage in cardBoxes )
      cardImage.Image = null;

   statusPictureBox.Image = null; // clear status image
   dealerTotalLabel.Text = string.Empty; // clear dealer total
   playerTotalLabel.Text = string.Empty; // clear player total

   // create a new, shuffled deck on the web-service host
   dealer.Shuffle();

   // deal two cards to player
   playersCards = dealer.DealCard(); // deal first card to player
   DisplayCard( 11, playersCards ); // display card
   card = dealer.DealCard(); // deal second card to player
   DisplayCard( 12, card ); // update GUI to display new card
   playersCards += '\t' + card; // add second card to player's hand
```

`BlackjackForm.cs`

(11 of 17)

The **Deal** button clears the `PictureBox`es and `Label`s for a new game.

Shuffling the deck and dealing two cards to each player.

Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 11 of 17.)

Outline

```

// deal two cards to dealer, only display face of first card
dealersCards = dealer.DealCard(); // deal first card to dealer
DisplayCard( 0, dealersCards ); // display card
card = dealer.DealCard(); // deal second card to dealer
DisplayCard( 1, string.Empty ); // display card face down
dealersCards += '\t' + card; // add second card to dealer's hand

stayButton.Enabled = true; // allow player to stay
hitButton.Enabled = true; // allow player to hit
dealButton.Enabled = false; // disable Deal Button

// determine the value of the two hands
int dealersTotal = dealer.GetHandValue( dealersCards );
int playersTotal = dealer.GetHandValue( playersCards );

// if hands equal 21, it is a push
if ( dealersTotal == playersTotal && dealersTotal == 21 )
   GameOver( GameStatus.PUSH );
else if ( dealersTotal == 21 ) // if dealer has 21, dealer wins
   GameOver( GameStatus.LOSE );
else if ( playersTotal == 21 ) // player has blackjack
   GameOver( GameStatus.BLACKJACK );
```

`BlackjackForm.cs`

(12 of 17)

Shuffling the deck and dealing two cards to each player.

Evaluating both the dealer’s and player’s hands.

Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 12 of 17.)

```
         // next dealer card has index 2 in cardBoxes
         currentDealerCard = 2;

         // next player card has index 13 in cardBoxes
         currentPlayerCard = 13;
      } // end method dealButton

      // deal another card to player
      private void hitButton_Click( object sender, EventArgs e )
      {
         string card = dealer.DealCard(); // deal new card
         playersCards += '\t' + card; // add new card to player's hand

         DisplayCard( currentPlayerCard, card ); // display card
         ++currentPlayerCard;

         // determine the value of the player's hand
         int total = dealer.GetHandValue( playersCards );
```

`BlackjackForm.cs`

(13 of 17)

Each time a player clicks **Hit**, the program deals the player one more card.

Evaluating the player’s hand, and having the dealer decide whether to draw a card.

Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 13 of 17.)

```
         // if player exceeds 21, house wins
         if ( total > 21 )
            GameOver( GameStatus.LOSE );
         else if ( total == 21 ) // if player has 21, dealer's turn
         {
            hitButton.Enabled = false;
            DealerPlay();
         } // end if
      } // end method hitButton_Click

      // play the dealer's hand after the player chooses to stay
      private void stayButton_Click( object sender, EventArgs e )
      {
         stayButton.Enabled = false; // disable Stay Button
         hitButton.Enabled = false; // disable Hit Button
         dealButton.Enabled = true; // enable Deal Button
         DealerPlay(); // player chose to stay, so play the dealer's hand
      } // end method stayButton_Click
   } // end class BlackjackForm
} // end namespace BlackjackClient
```

`BlackjackForm.cs`

(14 of 17)

Evaluating the player’s hand, and having the dealer decide whether to draw a card.

Clicking the **Stay** button disables the **Hit** and **Stay** buttons, then calls method `DealerPlay`.

Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 14 of 17.)

BlackjackForm.cs

(15 of 17)

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

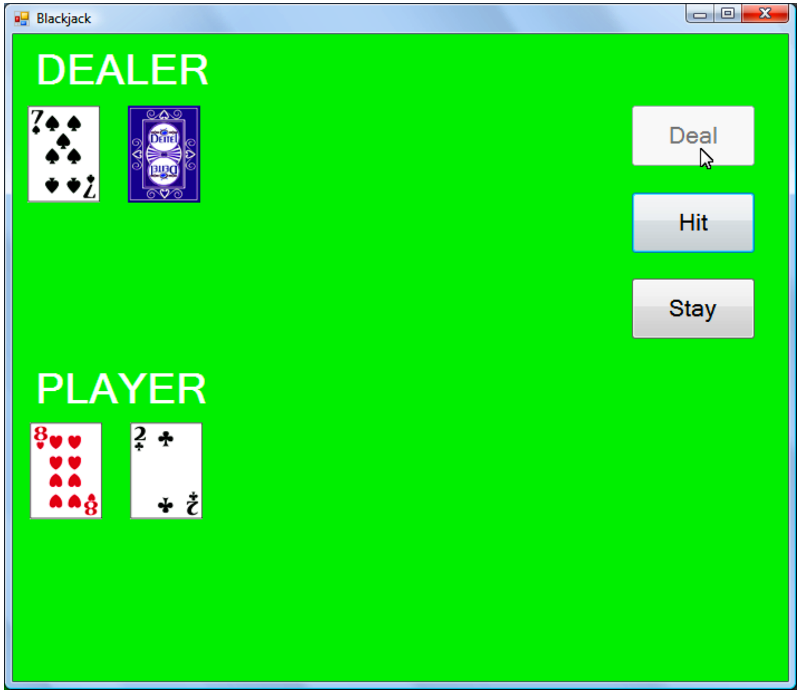
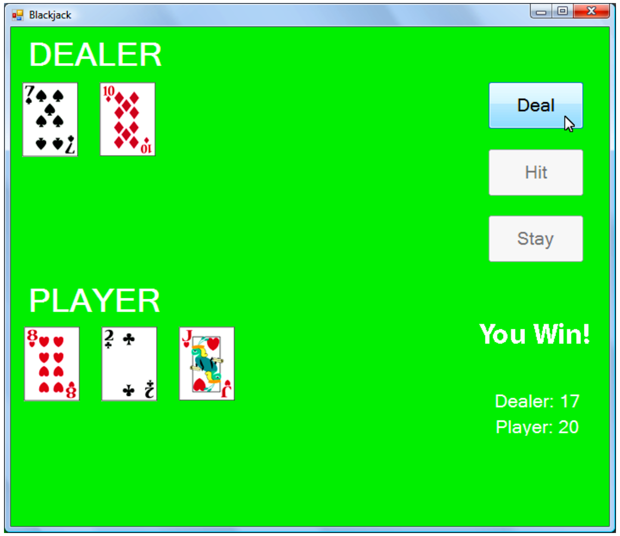


Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 15 of 17.)

BlackjackForm.cs

(16 of 17)

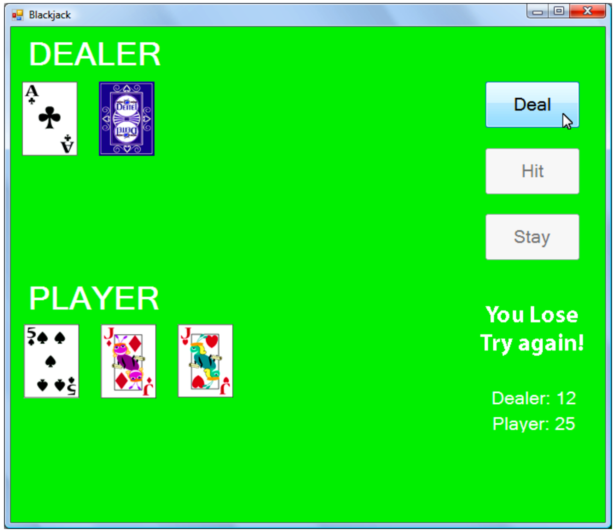
b) Cards after the player presses the Hit button once, then the Stay button. In this case, the player wins the game with a higher total than the dealer.



You Win!

Dealer: 17
Player: 20

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.



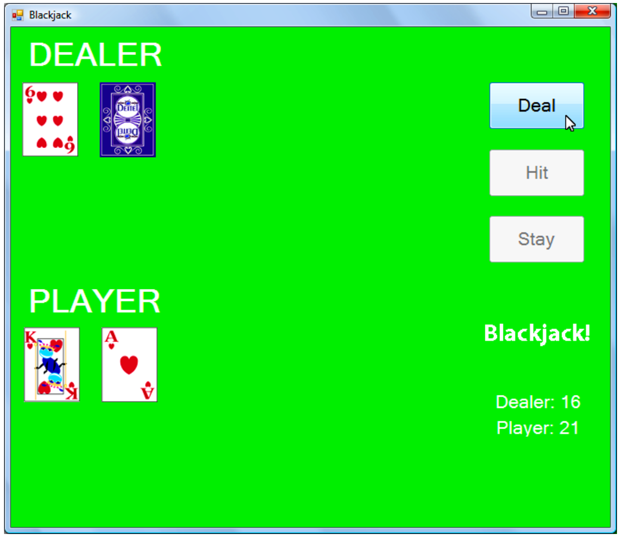
You Lose
Try again!

Dealer: 12
Player: 25

Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 16 of 17.)

Outline

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).



BlackjackForm.cs

(17 of 17)

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

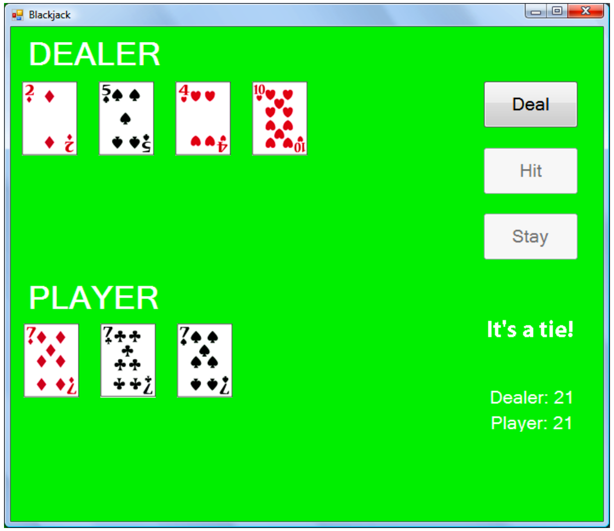


Fig. 15b.22 | Blackjack game that uses the `BlackjackService` web service. (Part 17 of 17.)

- You can easily use web services in ASP.NET web applications.
- Figure 15b.23 presents the interface for an airline reservation service.

IReservation
Service.cs

```
// Fig. 23.23: IReservationService.cs
// Airline reservation WCF web-service interface.
using System.ServiceModel;

[ServiceContract]
public interface IReservationService
{
   // reserves a seat
   [OperationContract]
   bool Reserve( string seatType, string classType );
} // end interface IReservationService
```

Fig. 15b.23 | Airline reservation WCF web-service interface.

- Add the `Tickets.mdf` database and corresponding LINQ to SQL classes to create a `DataContext` class.
- Figure 15b.24 presents the code-behind file for the web service..

ReservationService.cs

(1 of 2)

```
// Fig. 23.24: ReservationService.cs
// Airline reservation WCF web service.
using System.Linq;

public class ReservationService : IReservationService
{
   // create ticketsDB object to access Tickets database
   private TicketsDataContext ticketsDB = new TicketsDataContext();

   // checks database to determine whether matching seat is available
   public bool Reserve( string seatType, string classType )
   {
      //  LINQ query to find seats matching the parameters
      var result =
         from seat in ticketsDB.Seats
         where ( seat.Taken == false ) && ( seat.SeatType == seatType )
            && ( seat.SeatClass == classType )
         select seat;
```

Creating a `DataContext` object for database interaction.

Retrieving available seat numbers that match the query.

Fig. 15b.24 | Airline reservation WCF web service. (Part 1 of 2.)

Outline

ReservationService.cs

(2 of 2)

```

      // get first available seat
      Seat firstAvailableSeat = result.FirstOrDefault();

      // if seat is available seats, mark it as taken
      if ( firstAvailableSeat != null )
      {
         firstAvailableSeat.Taken = true; // mark the seat as taken
         ticketsDB.SubmitChanges(); // update
         return true; // seat was reserved
      } // end if

      return false; // no seat was reserved
   } // end method Reserve
} // end class ReservationService
```

The query result’s `FirstOrDefault` method returns the first available seat or a `null` value.

Reserving a seat and submitting changes to the database.

Fig. 15b.24 | Airline reservation WCF web service. (Part 2 of 2.)

Outline

- Figure 15b.25 presents the code for an ASP.NET page through which users can select seat types.
- Remember to add a service reference to the `ReservationService`.

`ReservationClient.aspx`

(1 of 3)

```
<%-- Fig. 23.25: ReservationClient.aspx                      --%>
<%-- Web Form that allows users to reserve seats on a plane. --%>
<%@ Page Language="C#" AutoEventWireup="true"
   CodeFile="ReservationClient.aspx.cs" Inherits="ReservationClient" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
   "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Ticket Reservation</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
```

Fig. 15b.25 | ASPX file that takes reservation information. (Part 1 of 3.)

ReservationClient.aspx

(2 of 3)

```
      <asp:Label ID="instructionsLabel" runat="server"
         Text="Please select the seat type and class to reserve:">
      </asp:Label>
      <br /><br />
      <%-- seat options --%>
      <asp:DropDownList ID="seatList" runat="server"
         Height="22px" Width="100px">
         <asp:ListItem>Aisle</asp:ListItem>
         <asp:ListItem>Middle</asp:ListItem>
         <asp:ListItem>Window</asp:ListItem>
      </asp:DropDownList>
      &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
      <%-- class options --%>
      <asp:DropDownList ID="classList" runat="server" Width="100px">
         <asp:ListItem>Economy</asp:ListItem>
```

A `DropDownList` displays the seat types from which users can select.

A `DropDownList` provides choices for the class type.

Fig. 15b.25 | ASPX file that takes reservation information. (Part 2 of 3.)

Reservation Client.aspx

(3 of 3)

```
         <asp:ListItem>First</asp:ListItem>
      </asp:DropDownList>
      &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
      <%-- submits selections to server --%>
      <asp:Button ID="reserveButton" runat="server" Height="24px"
         OnClick="reserveButton_Click"
         Text="Reserve" Width="102px" />
      <br /><br />
      <asp:Label ID="errorLabel" runat="server" ForeColor="#C00000"
         Height="19px" Width="343px"></asp:Label>
   </div>
   </form>
</body>
</html>
```

A DropDownList provides choices for the class type.

Fig. 15b.25 | ASPX file that takes reservation information. (Part 3 of 3.)

- Figure 15b.26 presents the code-behind file for the ASP.NET page.

ReservationClient.aspx.cs

(1 of 2)

```
// Fig. 23.26: ReservationClient.aspx.cs
// ReservationClient code-behind file.
using System;

public partial class ReservationClient : System.Web.UI.Page
{
   // object of proxy type used to connect to ReservationService
   private ServiceReference.ReservationServiceClient ticketAgent =
      new ServiceReference.ReservationServiceClient();

   // attempt to reserve the selected type of seat
   protected void reserveButton_Click( object sender, EventArgs e )
   {
      // if the ticket is reserved
      if ( ticketAgent.Reserve( seatList.SelectedItem.Text,
         classList.SelectedItem.Text ) )
      {
```

Creating a `ReservationServiceClient` proxy object.

Calling the web service's `Reserve` method and determining whether a seat was reserved.

Fig. 15b.26 | `ReservationClient` code-behind file. (Part 1 of 2.)

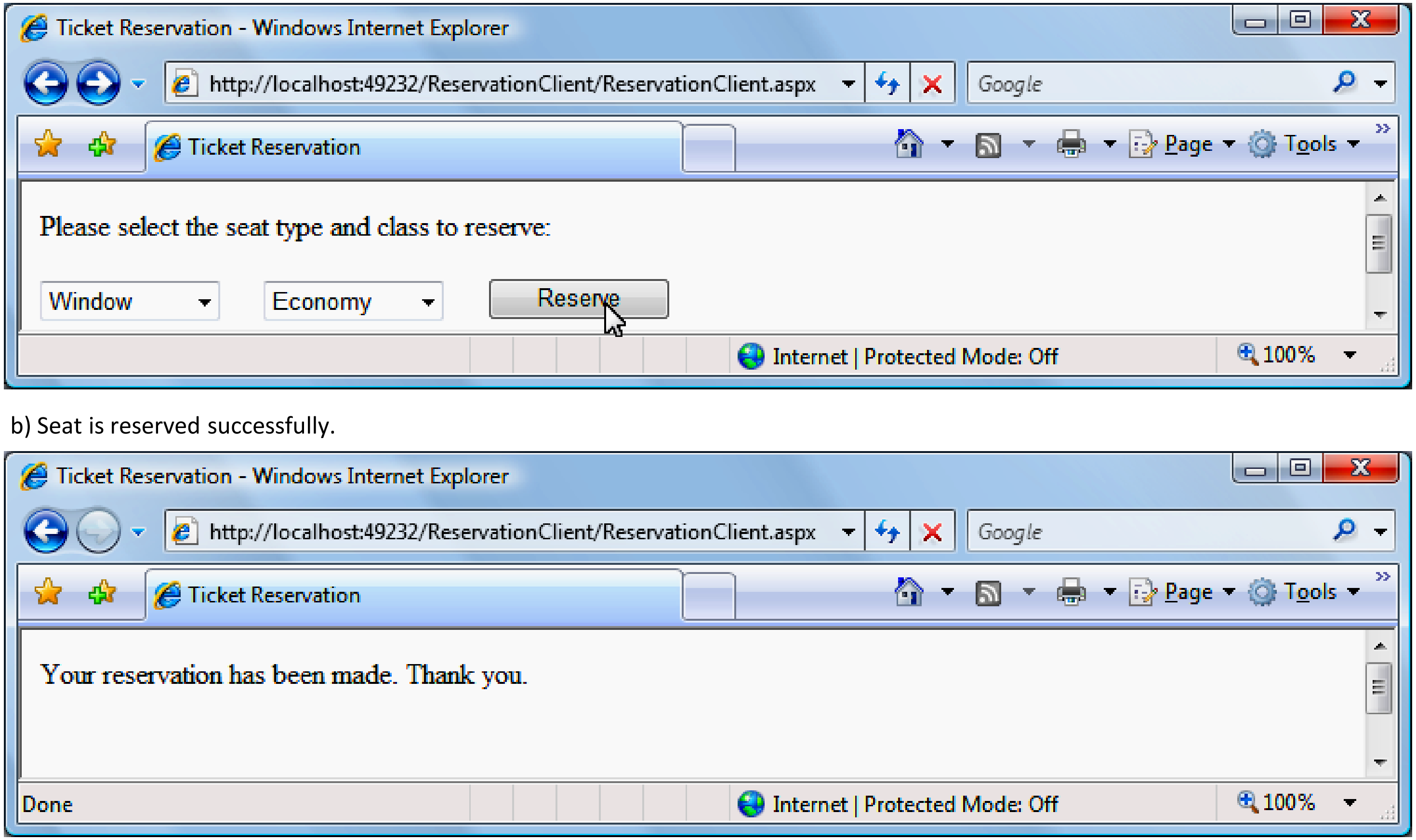
ReservationClient.aspx.cs

(2 of 2)

```
         // hide other controls
         instructionsLabel.Visible = false;
         seatList.Visible = false;
         classList.Visible = false;
         reserveButton.Visible = false;
         errorLabel.Visible = false;

         // display message indicating success
         Response.Write( "Your reservation has been made. Thank you." );
      } // end if
      else // service method returned false, so signal failure
      {
         // display message in the initially blank errorLabel
         errorLabel.Text = "This type of seat is not available. " +
            "Please modify your request and try again.";
      } // end else
   } // end method reserveButton_Click
} // end class ReservationClient
```

Fig. 15b.26 | ReservationClient code-behind file. (Part 2 of 2.)

10 Airline Reservation Web Service: Database Access and Invoking a Service from ASP.NET (Cont.)



a) Selecting a seat.

Ticket Reservation - Windows Internet Explorer

http://localhost:49232/ReservationClient/ReservationClient.aspx

Please select the seat type and class to reserve:

Window

Economy

b) Seat is reserved successfully.

Ticket Reservation - Windows Internet Explorer

http://localhost:49232/ReservationClient/ReservationClient.aspx

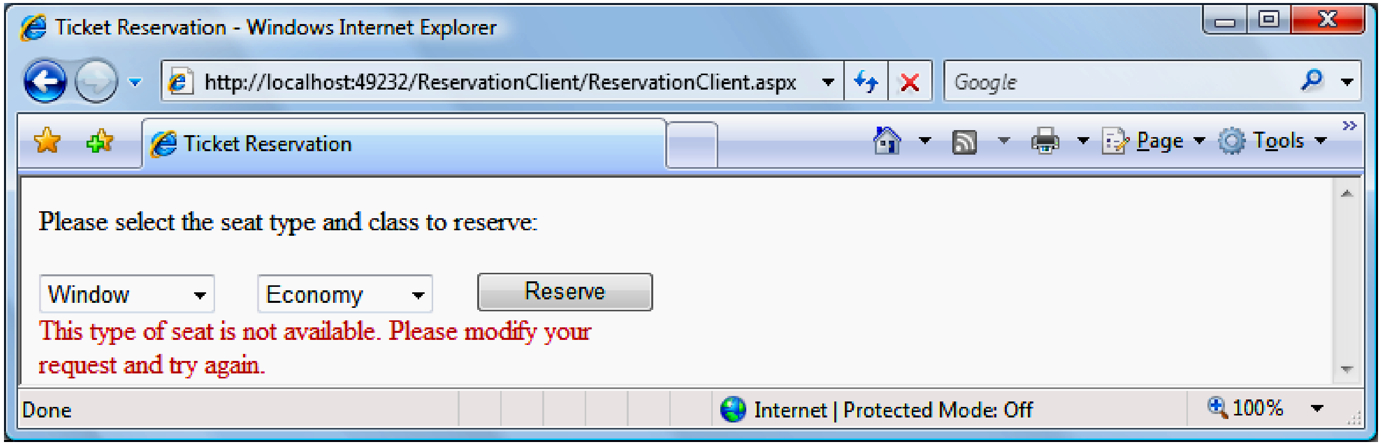
Your reservation has been made. Thank you.

Done

Fig. 15b.27 | Ticket reservation web-application sample execution. (Part 1 of 2.)

10 Airline Reservation Web Service: Database Access and Invoking a Service from ASP.NET (Cont.)

c) Attempting to reserve another seat.

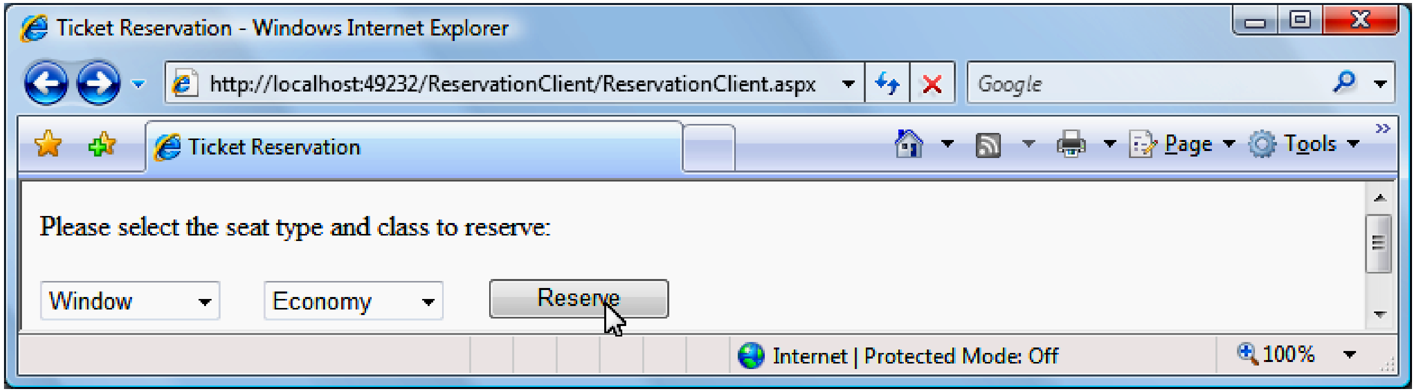

Ticket Reservation - Windows Internet Explorer
http://localhost:49232/ReservationClient/ReservationClient.aspx
Google
Ticket Reservation
Page
Tools
Done
Internet | Protected Mode: Off
100%

Please select the seat type and class to reserve:

Window ▾ Economy ▾ Reserve

This type of seat is not available. Please modify your request and try again.

d) No seats match the requested type and class.


Ticket Reservation - Windows Internet Explorer
http://localhost:49232/ReservationClient/ReservationClient.aspx
Google
Ticket Reservation
Page
Tools
Internet | Protected Mode: Off
100%

Please select the seat type and class to reserve:

Window ▾ Economy ▾ Reserve

Fig. 15b.27 | Ticket reservation web-application sample execution. (Part 2 of 2.)

11 Equation Generator: Returning User-Defined Types

This section presents an `EquationGenerator` web service.

The service generates random arithmetic equations of type `Equation`.

The client uses user-inputted information to request an equation.

The difficulty level is a `string` because variables for `UriTemplate` path segments must be of type `string`.

Equation.cs

(1 of 5)

```
// Fig. 23.28: Equation.cs
// Class Equation that contains information about an equation.
using System.Runtime.Serialization;

[DataContract]
public class Equation
{
   // automatic property to access the left operand
   [DataMember]
   private int Left { get; set; }

   // automatic property to access the right operand
   [DataMember]
   private int Right { get; set; }

   // automatic property to access the result of applying
   // an operation to the left and right operands
   [DataMember]
   private int Result { get; set; }

```

Fig. 15b.28 | Class `Equation` that contains information about an equation. (Part 1 of 5.)

Equation.cs

(2 of 5)

```
   // automatic property to access the operation
   [DataMember]
   private string Operation { get; set; }

   // required default constructor
   public Equation()
      : this( 0, 0, "add" )
   {
      // empty body
   } // end default constructor

   // three-argument constructor for class Equation
   public Equation( int leftValue, int rightValue, string type )
   {
      Left = leftValue;
      Right = rightValue;

```

The parameterless constructor calls the three-argument constructor with default values.

This constructor takes the left and right operands and the arithmetic operation as arguments.

Fig. 15b.28 | Class `Equation` that contains information about an equation. (Part 2 of 5.)

```
      switch ( type ) // perform appropriate operation
      {
         case "add": // addition
            Result = Left + Right;
            Operation = "+";
            break;
         case "subtract": // subtraction
            Result = Left - Right;
            Operation = "-";
            break;
         case "multiply": // multiplication
            Result = Left * Right;
            Operation = "*";
            break;
      } // end switch
   } // end three-argument constructor

   // return string representation of the Equation object
   public override string ToString()
   {
      return string.Format( "{0} {1} {2} = {4}", Left, Operation,
         Right, Result );
   } // end method ToString
```

`Equation.cs`

(3 of 5)

This constructor takes the left and right operands and the arithmetic operation as arguments.

Fig. 15b.28 | Class `Equation` that contains information about an equation. (Part 3 of 5.)

Equation.cs

(4 of 5)

```

   // property that returns a string representing left-hand side
   [DataMember]
   private string LeftHandSide
   {
      get
      {
         return string.Format( "{0} {1} {2}", Left, Operation,
            Right );
      } // end get
      set
      {
         // empty body
      } // end set
   } // end property LeftHandSide

```

Fig. 15b.28 | Class `Equation` that contains information about an equation. (Part 4 of 5.)

Equation.cs

(5 of 5)

```
   // property that returns a string representing right-hand side
   [DataMember]
   private string RightHandSide
   {
      get
      {
         return Result.ToString();
      } // end get
      set
      {
         // empty body
      } // end set
   } // end property RightHandSide
} // end class Equation
```

Fig. 15b.28 | Class `Equation` that contains information about an equation. (Part 5 of 5.)

Outline

- Figures 15b.29–15b.30 present the EquationGeneratorService's interface and class for creating randomly generated Equations.
- Modify the Web.config file to enable REST support as well.

IEquation GeneratorService .cs

```
// Fig. 23.29: 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;

[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
```

Defining the REST request for an equation with a certain operation and difficulty level.

Fig. 15b.29 | WCF REST service interface to create random equations based on a specified operation and difficulty level.

EquationGenerator Service.cs

(1 of 2)

```
// Fig. 22.30: EquationGeneratorService.cs
// WCF REST service to create random equations based on a
// specified operation and difficulty level.
using System;

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 =
```

`GenerateEquation`’s parameters represent the mathematical operation and the difficulty level.

Fig. 15b.30 | WCF REST service to create random equations based on a specified operation and difficulty level. (Part 1 of 2.)

EquationGenerator Service.cs

(2 of 2)

```
         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
```

The `Equation` is automatically serialized as an XML response.

Fig. 15b.30 | WCF REST service to create random equations based on a specified operation and difficulty level. (Part 2 of 2)

Outline

- The MathTutor application (Fig. 15b.31) uses the web service to create its Equation objects.

MathTutorForm.cs

(1 of 8)

```
// Fig. 23.31: MathTutorForm.cs
// Math tutor using EquationGeneratorServiceXML to create equations.
using System;
using System.Net;
using System.Windows.Forms;
using System.Xml.Linq;

namespace MathTutorXML
{
   public partial class MathTutorForm : Form
   {
      private string operation = "add"; // the default operation
      private int level = 1; // the default difficulty level
      private string leftHandSide; // the left side of the equation
      private int result; // the answer
      private XNamespace xmlNamespace =
         XNamespace.Get( "http://schemas.datacontract.org/2004/07/" );

      // object used to invoke service
      private WebClient service = new WebClient();
```

Defining the WebClient that is used to invoke the web service.

Fig. 15b.31 | Math tutor using EquationGeneratorServiceXML to create equations. (Part 1 of 8.)

```

      public MathTutorForm()
      {
         InitializeComponent();

         // add DownloadStringCompleted event handler to WebClient
         service.DownloadStringCompleted
            += new DownloadStringCompletedEventHandler(
            service_DownloadStringCompleted );
      } // end constructor

      // generates new equation when user clicks button
      private void generateButton_Click( object sender, EventArgs e )
      {
         // send request to EquationGeneratorServiceXML
         service.DownloadStringAsync( new Uri(
            "http://localhost:49732/EquationGeneratorServiceXML" +
            "/Service.svc/equation/" + operation + "/" + level ) );
      } // end method generateButton_Click

```

MathTutorForm.cs

(2 of 8)

Invoking the `EquationGeneratorService` asynchronously.

Fig. 15b.31 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 2 of 8.)

```
      // process web-service response
      private void service_DownloadStringCompleted(
         object sender, DownloadStringCompletedEventArgs e )
      {
         // check if any errors occurred in retrieving service data
         if ( e.Error == null )
         {
            // parse response and get LeftHandSide and Result values
            XDocument xmlResponse = XDocument.Parse( e.Result );
            leftHandSide = xmlResponse.Element(
               xmlNamespace + "Equation" ).Element(
               xmlNamespace + "LeftHandSide" ).Value;
            result = Convert.ToInt32( xmlResponse.Element(
               xmlNamespace + "Equation" ).Element(
               xmlNamespace + "Result" ).Value );

            // display left side of equation
            questionLabel.Text = leftHandSide;
            okButton.Enabled = true; // enable okButton
            answerTextBox.Enabled = true; // enable answerTextBox
         } // end if
      } // end method client_DownloadStringCompleted

```

MathTutorForm.cs

(3 of 8)

The `DownloadStringCompleted` event handler parses the XML response and displays the equation.

Fig. 15b.31 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 3 of 8.)

MathTutorForm.cs

(4 of 8)

```
      // check user's answer
      private void okButton_Click( object sender, EventArgs e )
      {
         if ( !string.IsNullOrEmpty( answerTextBox.Text ) )
         {
            // get user's answer
            int userAnswer = Convert.ToInt32( answerTextBox.Text );

            // determine whether user's answer is correct
            if ( result == userAnswer )
            {
               questionLabel.Text = string.Empty; // clear question
               answerTextBox.Clear(); // clear answer
               okButton.Enabled = false; // disable OK button
               MessageBox.Show( "Correct! Good job!", "Result" );
            } // end if
            else
            {
               MessageBox.Show( "Incorrect. Try again.", "Result" );
            } // end else
         } // end if
      } // end method okButton_Click
```

Checking whether the user provided the correct answer.

Fig. 15b.31 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 4 of 8.)

MathTutorForm.cs

(5 of 8)

```

      // set the operation to addition
      private void additionRadioButton_CheckedChanged( object sender,
         EventArgs e )
      {
         if ( additionRadioButton.Checked )
            operation = "add";
      } // end method additionRadioButton_CheckedChanged

      // set the operation to subtraction
      private void subtractionRadioButton_CheckedChanged( object sender,
         EventArgs e )
      {
         if ( subtractionRadioButton.Checked )
            operation = "subtract";
      } // end method subtractionRadioButton_CheckedChanged

```

Fig. 15b.31 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 5 of 8.)

```
   // set the operation to multiplication
   private void multiplicationRadioButton_CheckedChanged(
      object sender, EventArgs e )
   {
      if ( multiplicationRadioButton.Checked )
         operation = "multiply";
   } // end method multiplicationRadioButton_CheckedChanged

   // set difficulty level to 1
   private void levelOneRadioButton_CheckedChanged( object sender,
      EventArgs e )
   {
      if ( levelOneRadioButton.Checked )
         level = 1;
   } // end method levelOneRadioButton_CheckedChanged

   // set difficulty level to 2
   private void levelTwoRadioButton_CheckedChanged( object sender,
      EventArgs e )
   {
      if ( levelTwoRadioButton.Checked )
         level = 2;
   } // end method levelTwoRadioButton_CheckedChanged
```

`MathTutorForm.cs`

(6 of 8)

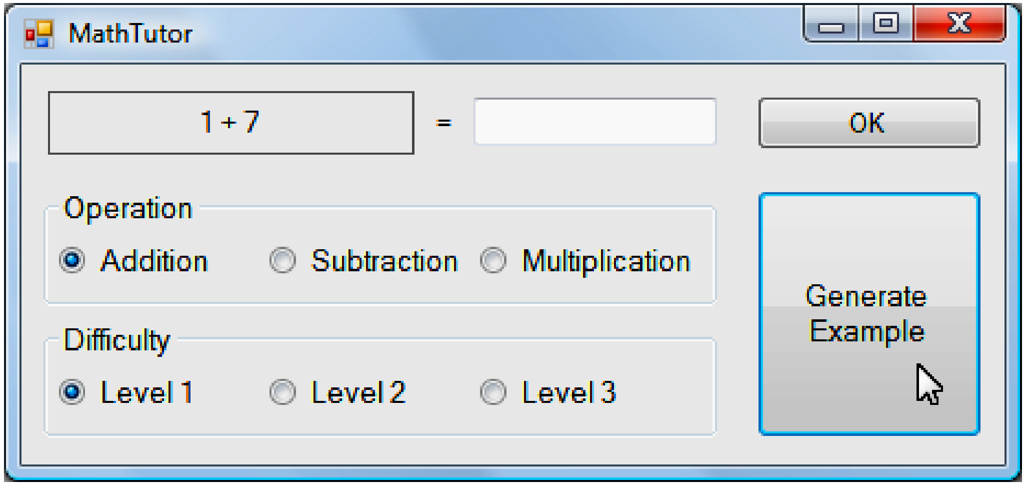
Fig. 15b.31 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 6 of 8.)

```

      // set difficulty level to 3
      private void levelThreeRadioButton_CheckedChanged( object sender,
         EventArgs e )
      {
         if ( levelThreeRadioButton.Checked )
            level = 3;
      } // end method levelThreeRadioButton_CheckedChanged
   } // end class MathTutorForm
} // end namespace MathTutorXML
```

MathTutorForm.cs

(7 of 8)

a) Generating a level 1 addition equation.



MathTutor

1 + 7 =

Operation: ● Addition ○ Subtraction ○ Multiplication

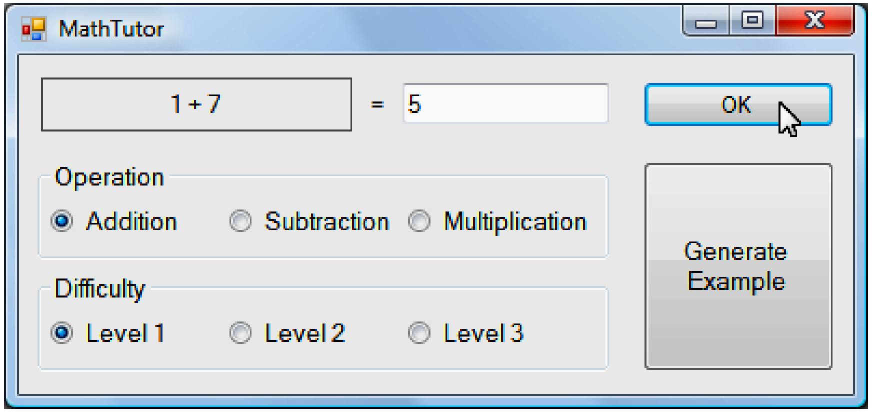
Difficulty: ● Level 1 ○ Level 2 ○ Level 3

Fig. 15b.31 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 7 of 8.)

MathTutorForm.cs

(8 of 8)

b) Answering the question incorrectly.



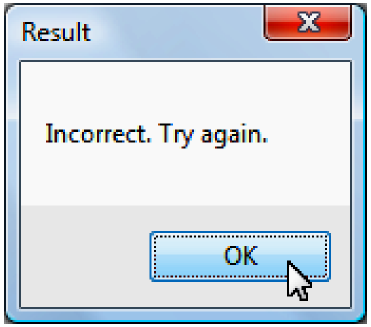
1 + 7 = 5

Operation

● Addition ○ Subtraction ○ Multiplication

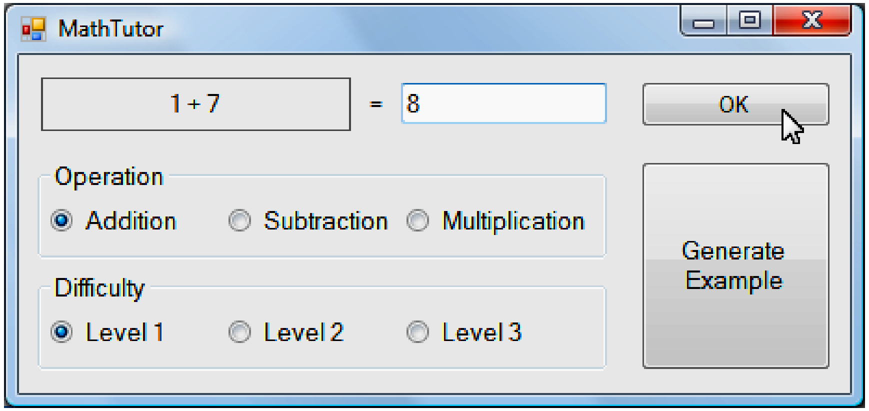
Difficulty

● Level 1 ○ Level 2 ○ Level 3



Incorrect. Try again.

c) Answering the question correctly.



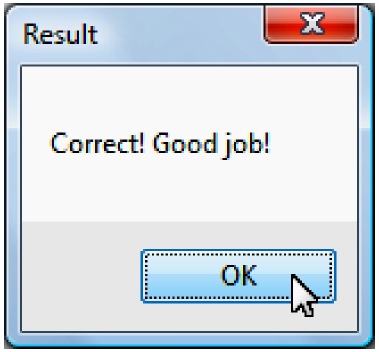
1 + 7 = 8

Operation

● Addition ○ Subtraction ○ Multiplication

Difficulty

● Level 1 ○ Level 2 ○ Level 3



Correct! Good job!

Fig. 15b.31 | Math tutor using `EquationGeneratorServiceXML` to create equations. (Part 8 of 8.)

- Figure 15b.32 is a modified `IEquationGeneratorService` interface that returns an Equation in JSON format.

IEquation GeneratorService .cs

```
// Fig. 23.32: 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;

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

Using the `ResponseFormat` property to specify a JSON response.

Fig. 15b.32 | WCF REST service interface to create random equations based on a specified operation and difficulty level.

Outline

- A modified `MathTutor` application (Fig. 15b.33) accesses the `EquationGenerator` web service.

`MathTutorForm.cs`

(1 of 8)

```
// Fig. 23.33: MathTutorForm.cs
// Math tutor using EquationGeneratorServiceJSON to create equations.
using System;
using System.IO;
using System.Net;
using System.Runtime.Serialization.Json;
using System.Text;
using System.Windows.Forms;

namespace MathTutorJSON
{
   public partial class MathTutorForm : Form
   {
      private string operation = "add"; // the default operation
      private int level = 1; // the default difficulty level
      private Equation currentEquation;   // represents the Equation

```

Fig. 15b.33 | Math tutor using `EquationGeneratorServiceJSON` to create equations. (Part 1 of 8.)

MathTutorForm.cs

(2 of 8)

```
      // object used to invoke service
      private WebClient service = new WebClient();

      public MathTutorForm()
      {
         InitializeComponent();

         // add DownloadStringCompleted event handler to WebClient
         service.DownloadStringCompleted
            += new DownloadStringCompletedEventHandler(
            service_DownloadStringCompleted );
      } // end constructor

      // generates new equation when user clicks button
      private void generateButton_Click( object sender, EventArgs e )
      {
         // send request to EquationGeneratorServiceJSON
         service.DownloadStringAsync( new Uri(
            "http://localhost:50103/EquationGeneratorServiceJSON" +
            "/Service.svc/equation/" + operation + "/" + level ) );
      } // end method generateButton_Click

```

Requesting an equation from the web service.

Fig. 15b.33 | Math tutor using `EquationGeneratorServiceJSON` to create equations. (Part 2 of 8.)

```
      // process web-service response
      private void service_DownloadStringCompleted(
         object sender, DownloadStringCompletedEventArgs e )
      {
         // check if any errors occurred in retrieving service data
         if ( e.Error == null )
         {
            // deserialize response into an Equation object
            DataContractJsonSerializer JSONSerializer =
               new DataContractJsonSerializer( typeof( Equation ) );
            currentEquation =
               ( Equation ) JSONSerializer.ReadObject( new
               MemoryStream( Encoding.Unicode.GetBytes( e.Result ) ) );

            // display left side of equation
            questionLabel.Text = currentEquation.LeftHandSide;
            okButton.Enabled = true; // enable okButton
            answerTextBox.Enabled = true; // enable answerTextBox
         } // end if
      } // end method client_DownloadStringCompleted

```

MathTutorForm.cs

(3 of 8)

Creating an object to deserialize `Equation`s from JSON format.

Converting JSON responses into `Equation` objects.

Fig. 15b.33 | Math tutor using `EquationGeneratorServiceJSON` to create equations. (Part 3 of 8.)

```
      // check user's answer
      private void okButton_Click( object sender, EventArgs e )
      {
         if ( !string.IsNullOrEmpty( answerTextBox.Text ) )
         {
            // determine whether user's answer is correct
            if ( currentEquation.Result ==
               Convert.ToInt32( answerTextBox.Text ) )
            {
               questionLabel.Text = string.Empty; // clear question
               answerTextBox.Clear(); // clear answer
               okButton.Enabled = false; // disable OK button
               MessageBox.Show( "Correct! Good job!", "Result" );
            } // end if
            else
            {
               MessageBox.Show( "Incorrect. Try again.", "Result" );
            } // end else
         } // end if
      } // end method okButton_Click

```

MathTutorForm.cs

(4 of 8)

Fig. 15b.33 | Math tutor using `EquationGeneratorServiceJSON` to create equations. (Part 4 of 8.)

```
      // set the operation to addition
      private void additionRadioButton_CheckedChanged( object sender,
         EventArgs e )
      {
         if ( additionRadioButton.Checked )
            operation = "add";
      } // end method additionRadioButton_CheckedChanged

      // set the operation to subtraction
      private void subtractionRadioButton_CheckedChanged( object sender,
         EventArgs e )
      {
         if ( subtractionRadioButton.Checked )
            operation = "subtract";
      } // end method subtractionRadioButton_CheckedChanged

      // set the operation to multiplication
      private void multiplicationRadioButton_CheckedChanged(
         object sender, EventArgs e )
      {
         if ( multiplicationRadioButton.Checked )
            operation = "multiply";
      } // end method multiplicationRadioButton_CheckedChanged
```

MathTutorForm.cs

(5 of 8)

Fig. 15b.33 | Math tutor using `EquationGeneratorServiceJSON` to create equations. (Part 5 of 8.)

MathTutorForm.cs

(6 of 8)

```
   // set difficulty level to 1
   private void levelOneRadioButton_CheckedChanged( object sender,
      EventArgs e )
   {
      if ( levelOneRadioButton.Checked )
         level = 1;
   } // end method levelOneRadioButton_CheckedChanged

   // set difficulty level to 2
   private void levelTwoRadioButton_CheckedChanged( object sender,
      EventArgs e )
   {
      if ( levelTwoRadioButton.Checked )
         level = 2;
   } // end method levelTwoRadioButton_CheckedChanged

```

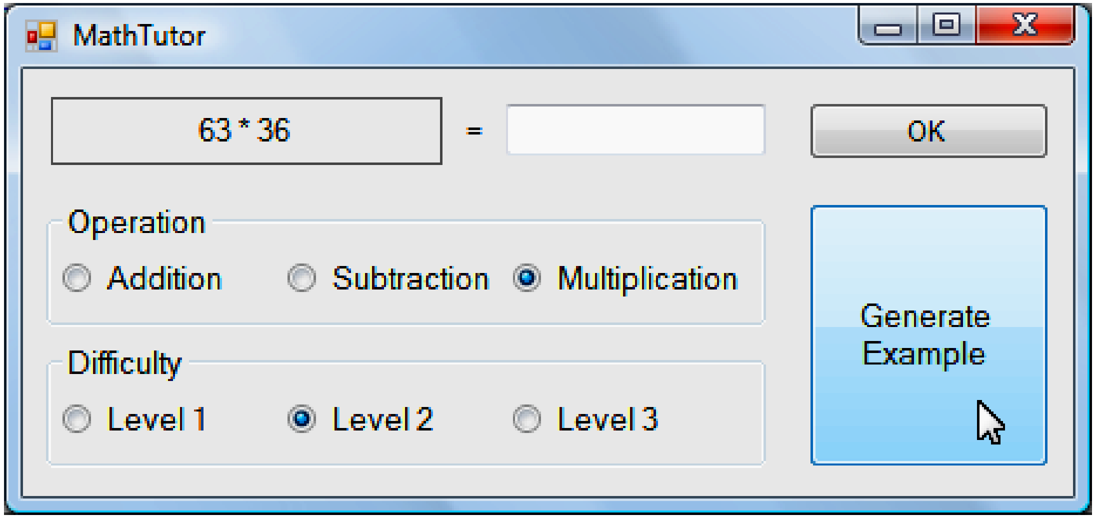
Fig. 15b.33 | Math tutor using `EquationGeneratorServiceJSON` to create equations. (Part 6 of 8.)

```
      // set difficulty level to 3
      private void levelThreeRadioButton_CheckedChanged( object sender,
         EventArgs e )
      {
         if ( levelThreeRadioButton.Checked )
            level = 3;
      } // end method levelThreeRadioButton_CheckedChanged
   } // end class MathTutorForm
} // end namespace MathTutorJSON
```

MathTutorForm.cs

(7 of 8)

a) Generating a level 2
multiplication equation.



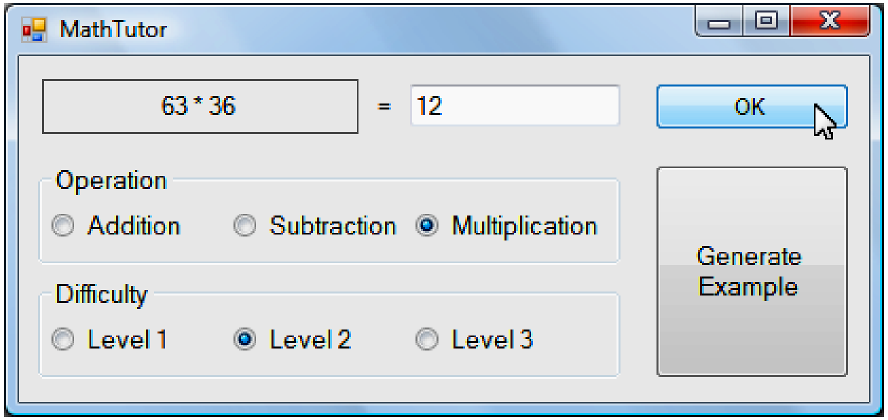
63 * 36 =

Operation
○ Addition ○ Subtraction ● Multiplication

Difficulty
○ Level 1 ● Level 2 ○ Level 3

Fig. 15b.33 | Math tutor using `EquationGeneratorServiceJSON` to create equations. (Part 7 of 8.)

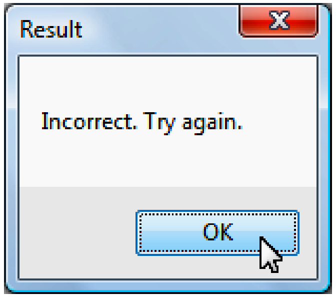
b) Answering the question incorrectly.



63 * 36 = 12

Operation
○ Addition ○ Subtraction ● Multiplication

Difficulty
○ Level 1 ● Level 2 ○ Level 3

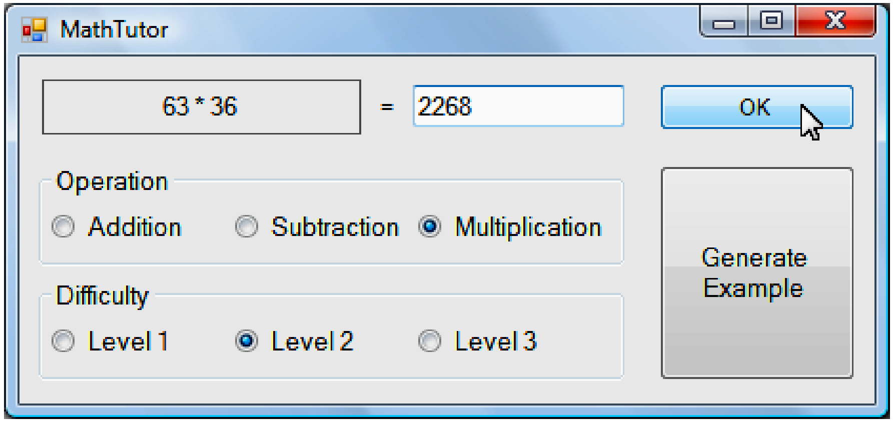


Incorrect. Try again.

MathTutorForm.cs

(8 of 8)

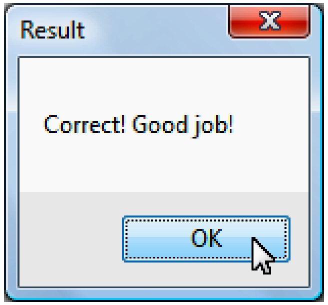
c) Answering the question correctly.



63 * 36 = 2268

Operation
○ Addition ○ Subtraction ● Multiplication

Difficulty
○ Level 1 ● Level 2 ○ Level 3



Correct! Good job!

Fig. 15b.33 | Math tutor using `EquationGeneratorServiceJSON` to create equations. (Part 8 of 8.)

- A JSON representation of an `Equation` object is shown in Fig. 15b.34.

`Equation.cs`

```
// Fig. 23.34: Equation.cs
// Equation class representing a JSON object.
using System;

namespace MathTutorJSON
{
   [Serializable]
   class Equation
   {
      public int Left = 0;
      public string LeftHandSide = null;
      public string Operation = null;
      public int Result = 0;
      public int Right = 0;
      public string RightHandSide = null;
   } // end class Equation
} // end namespace MathTutorJSON
```

Fig. 15b.34 | `Equation` class representing a JSON object.