Walkthrough:

# **Introduction**

This topic aims at understanding the need for a web service, the benefits of having a web service and how we can create a basic web service and consume it.

# **Background**

Connectivity between applications is very important. Connectivity in any case is very important for that matter but it specially is very important between applications. Connecting web application used to be a big challenge before the advent of technologies like SOAP (Simple Object Access Protocol). The reason it was so difficult was, there were so many technologies people were working in. The applications were hosted on different types of servers, etc. But these things should not be a barrier to facilitate the communication between applications. The only requirement was to have some standards to follow and standard ways of doing things so that the applications become capable of communicating with other applications irrespective of the technologies used.

# **SOAP and Web Services**

SOAP and XML created the solution for the problem that developers were facing before. SOAP is a standard XML based protocol that communicated over HTTP. We can think of SOAP as message format for sending messaged between applications using XML. It is independent of technology, platform and is extensible too. We have SOAP and XML to give us connectivity between applications. Does it mean that I have to write XML and SOAP specific things myself to facilitate this communications? I could do that but that will be very time consuming and sometimes error prone too.



Where does Web Services come in picture? Well, Web services is the mechanism that ASP.NET framework provides to make it easy for us to write code to facilitate connectivity between applications. As ASP.NET developer, If I need an application that will be used by many other applications then I can simply decide to write a web service for it and ASP.NET framework will take care of doing the low level SOAP and XML work for us.

The other side of the coin is, if our ASP.NET application wants to use a web service, i.e., an application that is providing me SOAP based interface for communication. So what we are going to do now is write a small Web service to see how we can have our application communication-ready for other applications and secondly, we will try to consume a webservice to understand how we can use other applications from our application.

#  **Creating and Using an ASP.NET Web Service**

In addition to letting you create Web pages, Microsoft Visual Studio also lets you create Web services that use ASP.NET XML. Creating a Web service in Visual Studio is similar to creating a Web page. You can also use the Microsoft Visual Web Developer Web development tool to reference and use Web services that are in a Visual Web Developer solution, on your local computer or in a local or external UDDI directory. In this walkthrough, you will create the Web service in one solution and use it in another.

Tasks illustrated in this walkthrough include:

* Creating a simple XML Web service in Visual Web Developer.
* Creating a separate Web site that uses the Web service.

# **Create The Web Service**

You will create a Web service that converts temperature from Fahrenheit to Celsius and vice versa. To create the Web service :

1. In Solution Explorer, right-click the Web site name (http://localhost/TemperatureWebService), and then click Add New Item.
2. Under Visual Studio installed templates, click Web Service, and then in the Name box, type Convert.
3. Make sure that the Place code in separate file check box is selected, and then click Add.

Visual Web Developer creates a new Web service that is made up of two files. The Convert.asmx file is the file that can be invoked to call Web service methods, and it points to the code for the Web service. The code itself is in a class file (Convert.vb, Convert.cs, or Convert.jsl, depending on the programming language) in the App\_Code folder. The code file contains a template for a Web service. The code file includes some code for a Web service method. You will create two methods in the Web service. The first method converts Fahrenheit temperatures to Celsius, and the second method converts Celsius temperatures to Fahrenheit.

1. Add the following code inside the class, after the HelloWorld method:

**[System.Web.Services.WebMethod()]**

**public double FahrenheitToCelsius(double Fahrenheit)**

**{**

 **return ((Fahrenheit - 32) \* 5) / 9;**

**}**

**[System.Web.Services.WebMethod()]**

**public double CelsiusToFahrenheit(double Celsius)**

**{**

 **return ((Celsius \* 9) / 5) + 32;**

**}**

Notice that the function names are preceded with an attribute ([System.Web.Services.WebMethod()] or <System.Web.Services.WebMethod()>) as part of the function declaration.

1. After you have entered the functions, save the file.

# **Test the Web service**

* In Solution Explorer, click Convert.asmx, and then press CTRL+F5.
* The Web service is invoked and a page appears in the browser that shows the methods that are exposed by the Web service.
* Click CelsiusToFahrenheit, which invokes that method.
* A page appears that prompts you for parameter values for the CelsiusToFahrenheit method.
* In the Celsius box, type 100, and then click Invoke.
* A new window appears that displays the XML that the Web service returns when the CelsiusToFahrenheit method is invoked. The value 212 appears in the XML.
* Close the browser that contains the method results.
* In the original browser, click Back to return to the list of methods.
* Click FahrenheitToCelsius and test to make sure that the method is returning the results that you expect.
* If you type 212, the FahrenheitToCelsius method will return 100.
* Close the browser.

# **Call the Web Service**

After you add a Web reference to the XML Web service to your project, you can use the server name to create a proxy object that allows you to call the methods in the XML Web service. the Web reference name (or server name) as the namespace, and the name of its .WSDL file (or the service name) as the proxy class.The following code example calls a Web service method to get a string value. Web service variable names and syntax are highlighted.

private void Call\_Web\_Service\_Method()

{

 ServerName.WebServiceName CallWebService =

 new ServerName.WebServiceName();

 String sGetValue = CallWebService.MethodName();

 Label1.Text = sGetValue;

}

Exposing Web Services

# **Exposing Web Services to Client Script**

The AJAX functionality of ASP.NET enables you to call ASP.NET Web services (.asmx files) from the browser by using client script. This enhances the user experience for the Web application. The page can call server-based methods without a postback and without refreshing the whole page, because only data is transferred between the browser and the Web server. This topic describes how to make a Web service available to JavaScript running in the browser.

ASP.NET automatically creates JavaScript proxy classes for Web services. Proxy classes are derived from the Sys.Net.WebServiceProxy class. You can call a Web service method by calling the corresponding method of the JavaScript proxy class. For more information, see Calling Web Services from Client Script.

# **Making Web Services Accessible from Script**

In order for a Web service to be accessed from script, it must be an .asmx Web service whose Web service class is qualified with the ScriptServiceAttribute attribute. Individual methods to be called from script must be qualified with the WebMethodAttribute attribute. The following example shows these attributes in Web service code.

[ScriptService]

public class SimpleWebService : System.Web.Services.WebService

{

 [WebMethod]

 public string EchoInput(String input)

 {

 // Method code goes here.

 }

}

To enable Web service calls from script, you must register the ScriptHandlerFactory HTTP handler in the application's Web.config file. The handler processes calls made from script to .asmx Web services. The following example shows the Web.config element for adding the handler.

<system.web>

 <httpHandlers>

 <remove verb="\*" path="\*.asmx"/>

 <add verb="\*" path="\*.asmx"

 type="System.Web.Script.Services.ScriptHandlerFactory"

 validate="false"/>

 </httpHandlers>

<system.web>

For Web service calls that are not issued from ASP.NET AJAX script, the ScriptHandlerFactory handler delegates the call to the default handler, which uses SOAP instead of JSON format. The delegation is performed automatically and you do not have to take any action unless you want to disable the use of the SOAP protocol for the Web services. In that case, you must enter the following configuration setting in the Web.config file.

<system.web>

 <webServices>

 <protocols>

 <clear/>

 </protocols>

 </webServices>

</system.web>

To enable an .asmx Web service to be called from client script in an ASP.NET Web page, you must add a ScriptManager control to the page. You reference the Web service by adding an asp:ServiceReference child element to the ScriptManager control and then setting the server reference path attribute to the URL of the Web service. The ServiceReference object instructs ASP.NET to generate a JavaScript proxy class for calling the specified Web service from client script.

The following example shows how to enable a Web service named SimpleWebService.asmx to be called from script in an ASP.NET Web page.

<asp:ScriptManager runat="server" ID="scriptManager">

 <Services>

 <asp:ServiceReference

 path="~/WebServices/SimpleWebService.asmx" />

 </Services>

</asp:ScriptManager>

The ServiceReference object can reference a Web service only in the same domain as the page. The Web service path can be relative, application relative, domain relative, or absolute. For absolute paths, you must make sure that the path is in the same domain.

When a page that contains this ScriptManager control is rendered, the page creates a JavaScript proxy class for the SimpleWebService.asmx Web service. The proxy class has methods that correspond to the Web methods in the SimpleWebService.asmx service. The page also contains JavaScript proxy classes that correspond to server data types that are used as input parameters or return values for the Web service methods. This enables you to write client script that initializes these parameters, and to pass them to the method call.