WEB SERVICES - EXAMPLES

Based on the web service architecture, we create the following two components as a part of web services implementation:

Service Provider or Publisher

This is the provider of the web service. The service provider implements the service and makes it available on the Internet or intranet.

We will write and publish a simple web service using .NET SDK.

Service Requestor or Consumer

This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

We will also write two web service requestors: one web-based consumer *ASP. NETapplication* and another Windows application-based consumer.

Given below is our first web service example which works as a service provider and exposes two methods *addandSayHello* as the web services to be used by applications. This is a standard template for a web service. .NET web services use the .asmx extension. Note that a method exposed as a web service has the WebMethod attribute. Save this file as FirstService.asmx in the IIS virtual directory *asexplainedinconfiguringIIS; forexample, c*: \MyWebSerces.

FirstService.asmx

```
<%@ WebService language = "C" class = "FirstService" %>

using System;
using System.Web.Services;
using System.Xml.Serialization;

[WebService(Namespace="http://localhost/MyWebServices/")]
public class FirstService : WebService
{
   [WebMethod]
   public int Add(int a, int b)
   {
      return a + b;
   }

   [WebMethod]
   public String SayHello()
   {
   return "Hello World";
   }
}
```

To test a web service, it must be published. A web service can be published either on an intranet or the Internet. We will publish this web service on IIS running on a local machine. Let us start with configuring the IIS.

- Open Start → Settings → Control Panel → Administrative tools → Internet Services Manager.
- Expand and right-click on the default web site; select New → Virtual Directory. The Virtual Directory Creation Wizard opens. Click Next.
- The "Virtual Directory Alias" screen opens. Type the virtual directory name. For example, MyWebServices. and click Next.
- The "Web Site Content Directory" screen opens.

- Enter the directory path name for the virtual directory. For example, c:\MyWebServices Click Next.
- The "Access Permission" screen opens. Change the settings as per your requirements. Let us keep the default settings for this exercise.
- Click the Next button. It completes the IIS configuration.
- Click Finish to complete the configuration.

To test whether the IIS has been configured properly, copy an HTML file *Forexample, x. html* in the virtual directory *C*:\MyWebServices created above. Now, open Internet Explorer and type http://localhost/MyWebServices/x.html. It should open the x.html file.

Note: If it does not work, try replacing the localhost with the IP address of your machine. If it still does not work, check whether IIS is running; you may need to reconfigure the IIS and the Virtual Directory.

To test this web service, copy FirstService.asmx in the IIS virtual directory created above *C*:\MyWebServices. Open the web service in Internet Explorer *http*: //*localhost*/*MyWebServices*/*FirstService*. *asmx*. It should open your web service page. The page should have links to two methods exposed as web services by our application. Congratulations! You have written your first web service!

Testing the Web Service

As we have just seen, writing web services is easy in the .NET Framework. Writing web service consumers is also easy in the .NET framework; however, it is a bit more involved. As said earlier, we will write two types of service consumers, one web-based and another Windows application-based consumer. Let us write our first web service consumer.

Web-Based Service Consumer

Write a web-based consumer as given below. Call it WebApp.aspx. Note that it is an ASP.NET application. Save this in the virtual directory of the web service *c*:\MyWebServices\WebApp. *axpx*.

This application has two text fields that are used to get numbers from the user to be added. It has one button, Execute, that when clicked gets the Add and SayHello web services.

```
WebApp.axpx
<%@ Page Language="C#" %>
<script runat="server">
   void runSrvice_Click(Object sender, EventArgs e){
      FirstService mySvc = new FirstService();
      Label1.Text = mySvc.SayHello();
      Label2.Text = mySvc.Add(Int32.Parse(txtNum1.Text),
Int32.Parse(txtNum2.Text)).ToString();
   }
</script>

<html>
   <head> </head>

   <body>
      <form runat="server">
         <p>
            <em>First Number to Add </em>:
            <asp:TextBox >4<  /asp:TextBox>
         </p>

         <p>
            <em>Second Number To Add </em>:
            <asp:TextBox >5</asp:TextBox>
         </p>

         <p>
            <strong><u>Web Service Result -</u></strong>
         </p>

         <p>
            <em>Hello world Service</em> :
            <asp:Label >Label< /asp:Label>
         </p>

         <p>
            <em>Add Service</em> :
            & <asp:Label >Label</asp:Label>
         </p>

         <p align="left">
            <asp:Button ></asp:Button>
         </p>

      </form>

   </body>
</html>
```

After the consumer is created, we need to create a proxy for the web service to be consumed. This work is done automatically by Visual Studio .NET for us when referencing a web service that has been added. Here are the steps to be followed:

- Create a proxy for the Web Service to be consumed. The proxy is created using the WSDL utility supplied with the .NET SDK. This utility extracts information from the Web Service and creates a proxy. The proxy is valid only for a particular Web Service. If you need to consume other Web Services, you need to create a proxy for this service as well. Visual Studio .NET creates a proxy automatically for you when the Web Service reference is added. Create a proxy for the Web Service using the WSDL utility supplied with the .NET SDK. It will create FirstSevice.cs file in the current directory. We need to compile it to create FirstService.dll *proxy* for the Web Service.

  ```
  c:> WSDL http://localhost/MyWebServices/FirstService.asmx?WSDL
  c:> csc /t:library FirstService.cs
  ```

- Put the compiled proxy in the bin directory of the virtual directory of the Web Service c: \MyWebServices\bin. Internet Information Services IIS looks for the proxy in this directory.

- Create the service consumer, in the same way we already did. Note that an object of the Web Service proxy is instantiated in the consumer. This proxy takes care of interacting with the service.

- Type the URL of the consumer in IE to test it *forexample, http: //localhost/MyWebServices/WebApp. aspx*.

Windows Application-Based Web Service Consumer

Writing a Windows application-based web service consumer is the same as writing any other Windows application. You only need to create the proxy *whichwehavealreadydone* and reference this proxy when compiling the application. Following is our Windows application that uses the web service. This application creates a web service object *ofcourse, proxy* and calls the SayHello, and Add methods on it.

WinApp.cs

```
using System;
using System.IO;

namespace SvcConsumer {
   class SvcEater {

      public static void Main(String[] args) {
         FirstService mySvc = new FirstService();
         Console.WriteLine("Calling Hello World Service: " + mySvc.SayHello());
         Console.WriteLine("Calling Add(2, 3) Service: " + mySvc.Add(2, 3).ToString());
      }
   }
```

```
}
```

Compile it using `c:\>csc /r:FirstService.dll WinApp.cs`. It will create WinApp.exe. Run it to test the application and the web service.

Now, the question arises: How can you be sure that this application is actually calling the web service?

It is simple to test. Stop your Web server so that the Web Service cannot be contacted. Now, run the WinApp application. It will fire a run-time exception. Now, start the web server again. It should work.