**Introduction**

We can now use ASP.NET to create Web Services based on industrial standards including XML, SOAP and WSDL.

A Web Service is a software program that uses XML to exchange information with other software via common internet protocols. In a simple sense, Web Services are a way for interacting with objects over the Internet.

A web service is

* Language Independent**.**
* Protocol Independent**.**
* Platform Independent**.**
* It assumes a stateless service architecture.
* Scalable (e.g. multiplying two numbers together to an entire customer-relationship management system).
* Programmable (encapsulates a task).
* Based on XML (open, text-based standard).
* Self-describing (metadata for access and use).
* Discoverable (search and locate in registries)- ability of applications and developers to search for and locate desired Web services through registries. This is based on UDDI.

**Web Service History**

* Microsoft coined the term "Web services" in June 2000, when the company introduced Web services as a key component of its .Net initiative, a broad new vision for embracing the Internet in the development, engineering and use of software.
* As others began to investigate Web services, it became clear that the technology could revolutionize (be the next stage in) distributed computing.
* Web services encom a set of related standards that can enable any two computers to communicate and exchange data via a network, such as the Internet.
* The primary standard used in Web services is the Extensible Markup Language (XML) developed by the World Wide Web Consortium (W3C).
* Developers use XML tags to describe individual pieces of data, forming XML documents, which are text-based and can be processed on any platform.
* XML provides the foundation for many core Web services standards (SOAP, WSDL, and UDDI) and vocabularies (XML-based markup for a specific industry or purpose).
* Almost every type of business can benefit from Web services such as expediting software development, integrating applications and databases, and automating transactions with suppliers, partners, and clients.

**Key Web Service Technologies**

* **XML**- Describes only data. So, any application that understands XML-regardless of the application's programming language or platform-has the ability to format XML in a variety of ways (well-formed or valid).
* **SOAP**- Provides a communication mechanism between services and applications.
* **WSDL**- Offers a uniform method of describing web services to other programs.
* **UDDI**- Enables the creation of searchable Web services registries.

When these technologies are deployed together, they allow developers to package applications as services and publish those services on a network.

**Web services advantages**

* Use open, text-based standards, which enable components written in various languages and for different platforms to communicate.
* Promote a modular approach to programming, so multiple organizations can communicate with the same Web service.
* Comparatively easy and inexpensive to implement, because they employ an existing infrastructure and because most applications can be repackaged as Web services.
* Significantly reduce the costs of enterprise application (EAI) integration and B2B communications.
* Implemented incrementally, rather than all at once which lessens the cost and reduces the organizational disruption from an abrupt switch in technologies.
* The Web Services Interoperability Organization (WS-I) consisting of over 100 vendors promotes interoperability.

**Web Services Limitations**

* SOAP, WSDL, UDDI- require further development.
* Interoperability.
* Royalty fees.
* Too slow for use in high-performance situations.
* Increase traffic on networks.
* The lack of security standards for Web services.
* The standard procedure for describing the quality (i.e. levels of performance, reliability, security etc.) of particular Web services â€“ management of Web services.
* The standards that drive Web services are still in draft form (always will be in refinement).
* Some vendors want to retain their intellectual property rights to certain Web services standards.

**Web Service Example**

A web service can perform almost any kind of task.

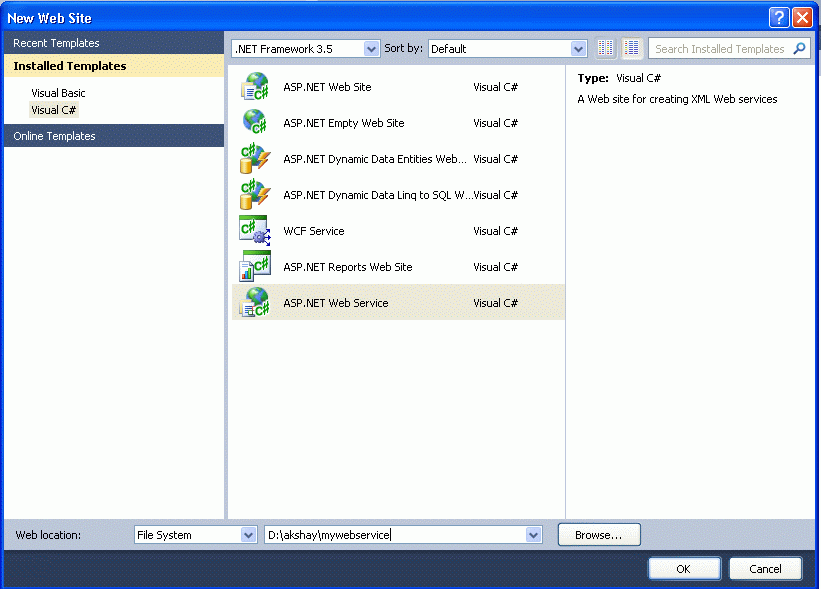
* **Web Portal-**A web portal might obtain top news headlines from an Associated press web service.
* **Weather Reporting-** You can use Weather Reporting web service to display weather information in your personal website.
* **Stock Quote-** You can display latest update of Share market with Stock Quote on your web site.
* **News Headline:** You can display latest news update by using News Headline Web Service in your website.
* You can make your own web service and let others use it. For example you can make Free SMS Sending Service with footer with your companies advertisement, so whosoever uses this service indirectly advertises your company. You can apply your ideas in N no. of ways to take advantage of it.

**Example of Creating Web Service in .Net**

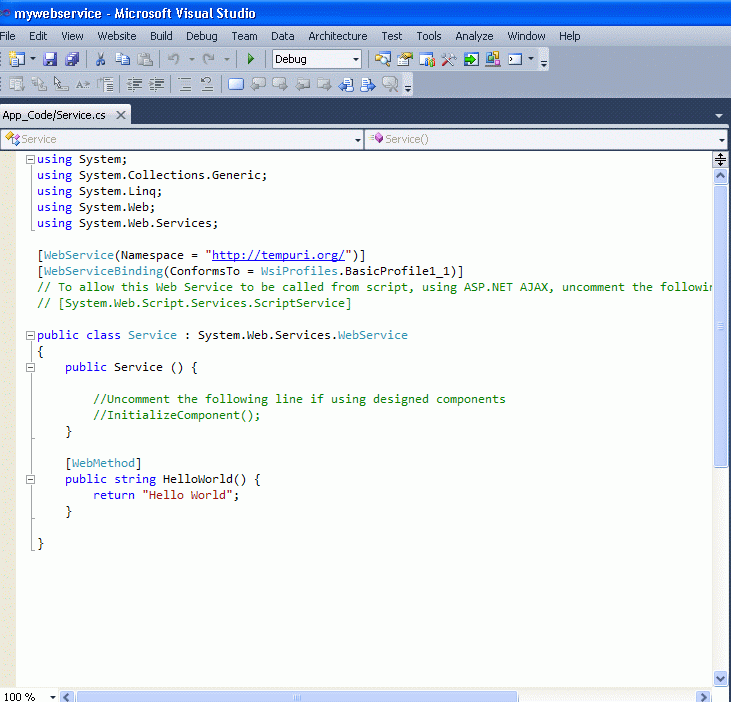
Here are samples codes which I use to create and consume ASP.NET Web Service:

* **Step 1- Create the ASP.NET Web Service Source File**

Open Visual Studio 2010 and create  a new web site.->Select .Net Framework 3.5. ->Select ASP.NET Web Service page -> Then, you have to give the name of your service. In this example I am giving it's name "**mywebservice**". Then Click the ok Button. A screen-shot of these activity is given below.



* **Step 2**- click on the "ok" button; you will see the following window.



Here (in the above figure), you will note that there is predefined method "**HelloWorld**" which returns the string "**Hello World**". You can use your own method and can perform various operations. Here I made a simple method which returns the multiplication of two numbers using the code.

**Service.cs**

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Web;  
using System.Web.Services;  
[WebService(Namespace = "http://tempuri.org/")]  
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]  
// To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.   
// [System.Web.Script.Services.ScriptService]   
public class Service : System.Web.Services.WebService  
{  
    [WebMethod]  
    public int Multiplication(int a,int b)  
    {  
        return (a\*b);  
    }  
}

**Before  Debugging the above Web Service see some important term**

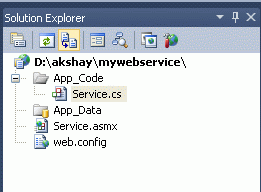
using System.Web.Services;

This directive allows you to refer to objects in the System.Web.Services namespace without having to fully qualify the request. This statement is optional, but if it is not included, every reference to an object in this namespace must be fully qualified. An example is the next line, which is our class declaration. With the using statement, it looks as follows in C#:

**The  [WebMethod] attribute**

The Service class exposes a single method, the public method **Multiplication**, which takes two integer arguments and returns the multiplication of two number as integer. To expose a method as a part of a web service, you must decorate it with the WebMethod attribute, which tells the compiler to treat it as such. Any method marked with the WebMethod attribute must be defined as public. Class methods exposed as web services follow the same object-oriented rules as any other class, and therefore methods marked private, protected, or internal are not accessible and will return an error if you attempt to expose them using the WebMethod attribute.

In the Solution Explorer you will see



**Service.asmx-**which contains the following code:

<%@ WebService Language="C#" CodeBehind="~/App\_Code/Service.cs" Class="Service" %>

The page directive WebService is required and the class is the name of the .NET Class to expose theWeb Service, each method exposes as Web Service Class Method need to have a declarative attribute statement.

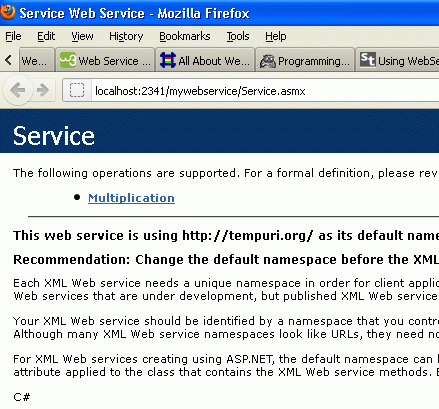
The WebService directive is similar to the Page directive that begins most .aspx pages. For the**Multiplication** web service to work, you must assign values to two WebService directive attributes: Language and Class.

The required Language attribute lets .NET know which programming language the class has been written in. As you might guess, the acceptable values for the language attribute are currently C#, VB, and JS for JScript.NET.

The Class attribute, also required, tells ASP.NET the name of the class to expose as a web service. Because a web service application can comprise multiple classes, some of which may not be web services, you must tell .NET which class to expose, a step analogous to declaring a Main() method to indicate the entry point of a .NET console application or component. Note that even if your web service contains only one class, setting this attribute is required.

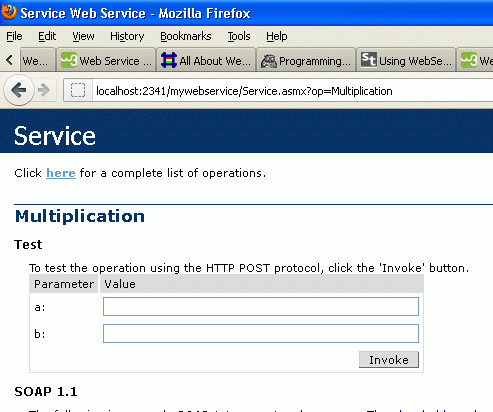
Now back to the our web service.

* **Step 3**- Build Web Service and Run the Web Service for testing by pressing F5 function key.

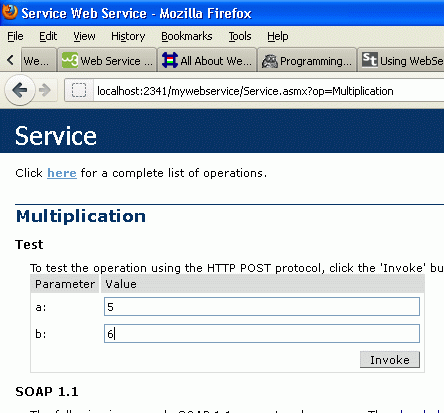


**Copy the url of this web service for further use.**

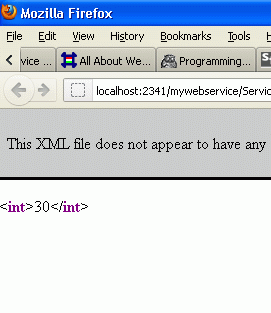
Click on the Mutiplication button to test the web service.



Enter the value of a and b.



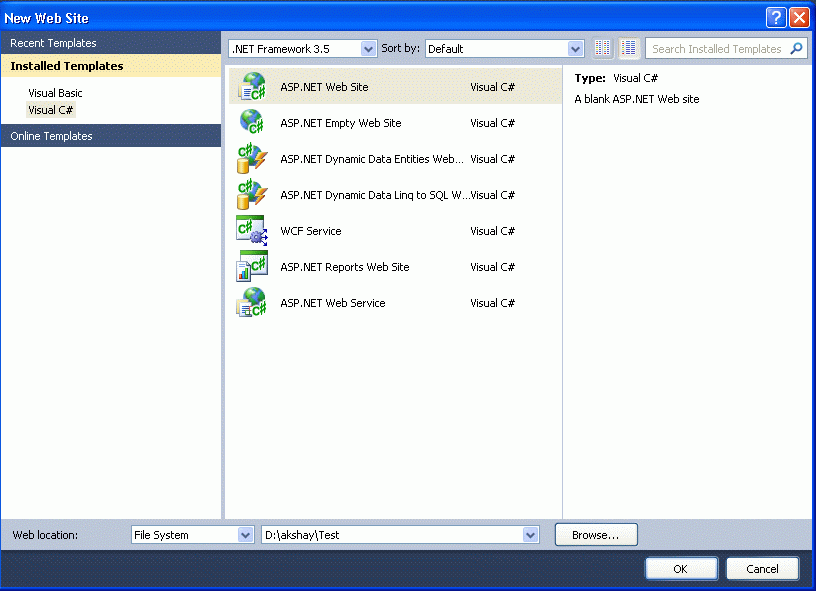
By pressing the "Invoke" button a XML file is generated.



Now our web service is ready to use; we just need to create a new web site to consume the web service.

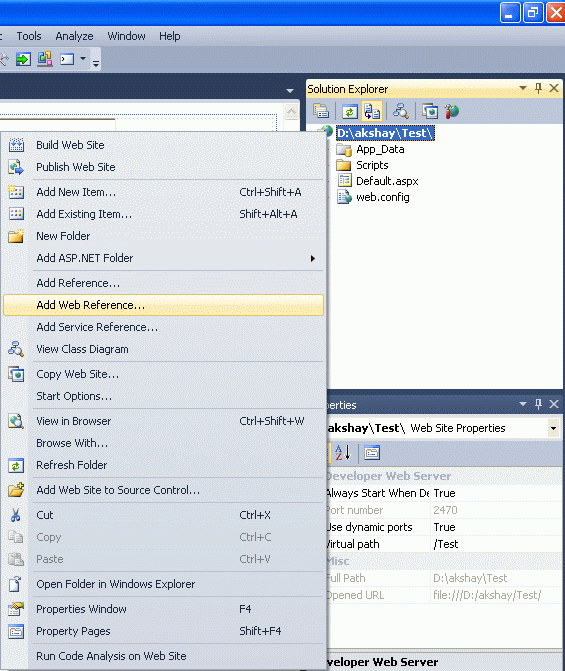
**Example of Testing Web Service in .Net.**

* **Step 5-**Create a Test Web Site by File > New > Web Site > Asp.net Web Site

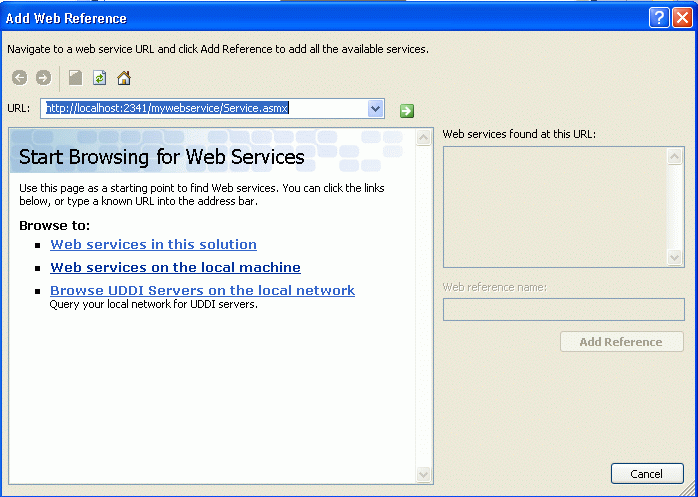


Name the web site, for example here I have choosen the name "Test".

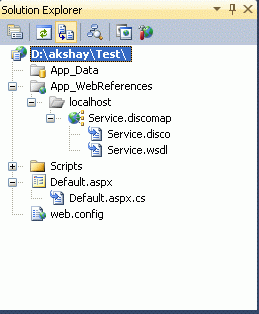
* **Step 6**- Right-click Solution Explorer and choose "Add Web Reference".



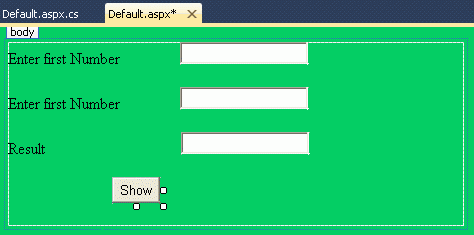
* **Step 7-**Paste the url of the web service and click on 'Go' button and then 'Add reference'.



* **Step 8-** Now your web service is ready to use in the Solution Explorer you will see.



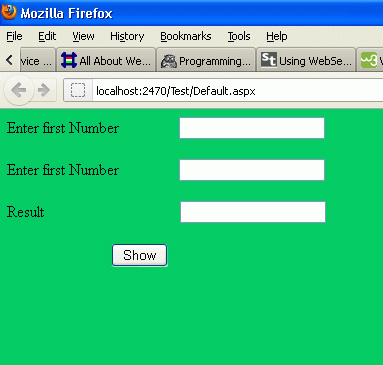
* **Step 9**- Go to the design of the Default.aspx page; drag and drop three Textboxes and one button.



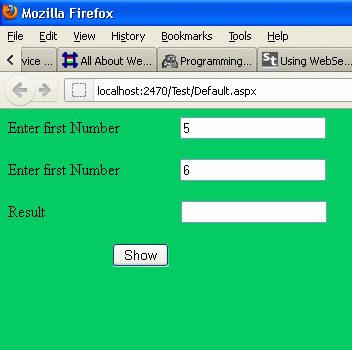
* **step 10-**Go to Default.cs page and on the button click event use the following code.

    protected void Button1\_Click(object sender, EventArgs e)  
    {  
        localhost.Service mys = new localhost.Service();   // you need to create the object of the web service   
        int a = Convert.ToInt32(TextBox1.Text);  
        int b = Convert.ToInt32(TextBox2.Text);  
        int c = mys.Multiplication(a, b);  
        TextBox3.Text = c.ToString();  
    }

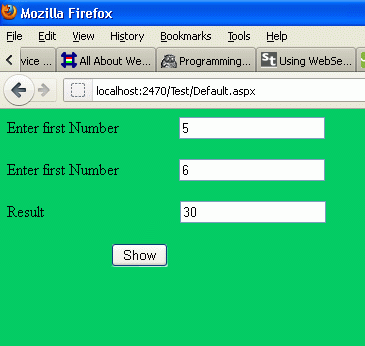
* **Step 11**- After pressing the F5 function key to run the website, you will see:



Enter the number.



Press the show button.



**Proxy Class**

Definition

Provide a surrogate or placeholder for another object to control access to it.

Frequency of use:

Medium high

UML class diagram

Participants

    The classes and objects participating in this pattern are:

* **Proxy**  **(MathProxy)**
  + maintains a reference that lets the proxy access the real subject. Proxy may refer to a Subject if the RealSubject and Subject interfaces are the same.
  + provides an interface identical to Subject's so that a proxy can be substituted for for the real subject.
  + controls access to the real subject and may be responsible for creating and deleting it.
  + other responsibilites depend on the kind of proxy:
    - *remote proxies* are responsible for encoding a request and its arguments and for sending the encoded request to the real subject in a different address space.
    - *virtual proxies* may cache additional information about the real subject so that they can postpone accessing it. For example, the ImageProxy from the Motivation caches the real images's extent.
    - *protection proxies* check that the caller has the access permissions required to perform a request.
* **Subject**  **(IMath)**
  + defines the common interface for RealSubject and Proxy so that a Proxy can be used anywhere a RealSubject is expected.
* **RealSubject**  **(Math)**
  + defines the real object that the proxy represents.

Structural code in C#

This structural code demonstrates the Proxy pattern which provides a representative object (proxy) that controls access to another similar object.

2. using System;
4. namespace DoFactory.GangOfFour.Proxy.Structural
5. {
6. /// <summary>
7. /// MainApp startup class for Structural
8. /// Proxy Design Pattern.
9. /// </summary>
10. class MainApp
11. {
12. /// <summary>
13. /// Entry point into console application.
14. /// </summary>
15. static void Main()
16. {
17. // Create proxy and request a service
18. Proxy proxy = new Proxy();
19. proxy.Request();
21. // Wait for user
22. Console.ReadKey();
23. }
24. }
26. /// <summary>
27. /// The 'Subject' abstract class
28. /// </summary>
29. abstract class Subject
30. {
31. public abstract void Request();
32. }
34. /// <summary>
35. /// The 'RealSubject' class
36. /// </summary>
37. class RealSubject : Subject
38. {
39. public override void Request()
40. {
41. Console.WriteLine("Called RealSubject.Request()");
42. }
43. }
45. /// <summary>
46. /// The 'Proxy' class
47. /// </summary>
48. class Proxy : Subject
49. {
50. private RealSubject \_realSubject;
52. public override void Request()
53. {
54. // Use 'lazy initialization'
55. if (\_realSubject == null)
56. {
57. \_realSubject = new RealSubject();
58. }
60. \_realSubject.Request();
61. }
62. }
63. }

Output

Called RealSubject.Request()

Real-world code in C#

This real-world code demonstrates the Proxy pattern for a Math object represented by a MathProxy object.

2. using System;
4. namespace DoFactory.GangOfFour.Proxy.RealWorld
5. {
6. /// <summary>
7. /// MainApp startup class for Real-World
8. /// Proxy Design Pattern.
9. /// </summary>
10. class MainApp
11. {
12. /// <summary>
13. /// Entry point into console application.
14. /// </summary>
15. static void Main()
16. {
17. // Create math proxy
18. MathProxy proxy = new MathProxy();
20. // Do the math
21. Console.WriteLine("4 + 2 = " + proxy.Add(4, 2));
22. Console.WriteLine("4 - 2 = " + proxy.Sub(4, 2));
23. Console.WriteLine("4 \* 2 = " + proxy.Mul(4, 2));
24. Console.WriteLine("4 / 2 = " + proxy.Div(4, 2));
26. // Wait for user
27. Console.ReadKey();
28. }
29. }
31. /// <summary>
32. /// The 'Subject interface
33. /// </summary>
34. public interface IMath
35. {
36. double Add(double x, double y);
37. double Sub(double x, double y);
38. double Mul(double x, double y);
39. double Div(double x, double y);
40. }
42. /// <summary>
43. /// The 'RealSubject' class
44. /// </summary>
45. class Math : IMath
46. {
47. public double Add(double x, double y) { return x + y; }
48. public double Sub(double x, double y) { return x - y; }
49. public double Mul(double x, double y) { return x \* y; }
50. public double Div(double x, double y) { return x / y; }
51. }
53. /// <summary>
54. /// The 'Proxy Object' class
55. /// </summary>
56. class MathProxy : IMath
57. {
58. private Math \_math = new Math();
60. public double Add(double x, double y)
61. {
62. return \_math.Add(x, y);
63. }
64. public double Sub(double x, double y)
65. {
66. return \_math.Sub(x, y);
67. }
68. public double Mul(double x, double y)
69. {
70. return \_math.Mul(x, y);
71. }
72. public double Div(double x, double y)
73. {
74. return \_math.Div(x, y);
75. }
76. }
77. }

Output

4 + 2 = 6  
4 - 2 = 2  
4 \* 2 = 8  
4 / 2 = 2