SOAP Web Services: Create Once, Consume Everywhere

By **Mohamad K. Ayyash** | 15 Jul 2010

.NET3.0 C#3.0 IIS7 .NET3.5 VS2008 C# ASP.NET XML Java Windows , +

A detailed development of a simple ASP.NET web service, in addition to configuring IIS server to host the service, finally creating an ASP.NET, Java and PHP web clients that consume the service.

Introduction

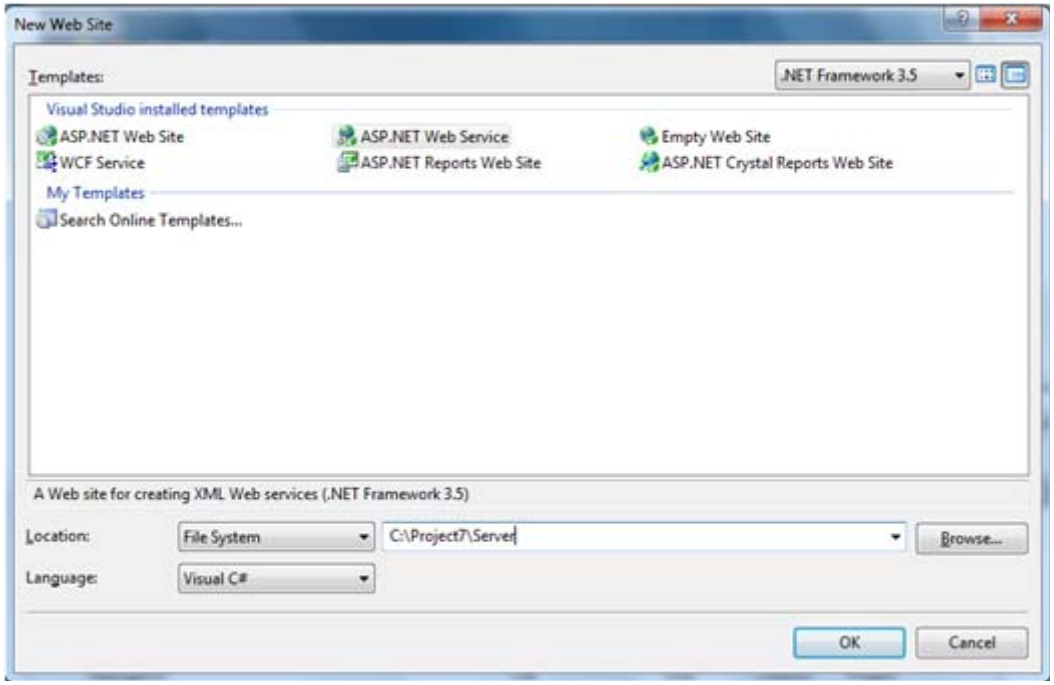
This tutorial aims to provide a practical introduction to web services given their increased importance in software engineering. In short, web services offer clients the ability to remotely execute certain methods thus eliminating the need for downloading separate API. The communication medium between clients and servers is XML messages that comply with the SOAP protocol. The use of XML allows systems running different operating systems and programming languages to communicate by converting XML requests and responses into their corresponding language equivalents. Fortunately, programming languages like C# (or any .NET language), Java, PHP, etc. support both clients and servers with abstraction layers called proxies that eliminate the need to parse XML SOAP messages. In this tutorial, I'll show you how to create a Web Service in ASP.NET and how to register this service in IIS server 7.0; finally I'll demonstrate the language neutrality of web services by developing three client applications that consume this service in ASP.NET, Java, and PHP.

Creating the ASP.NET Web Service

Launch Visual Studio, and then go to File / New / Web Site...

Choose ASP.NET Web Service as the template and name your project: *Server*.

Throughout this project, I'll use C:\Project7 as my default folder.



See Also...

Announcements

Go to the *Service.cs* file and create the four needed methods by replacing:

Collapse

```
[WebMethod]
public string HelloWorld() {
return "Hello World";
```

with:

Collapse

```
[[WebMethod]
    public int Add(int x, int y)
    {
        return x + y;
    }
    [WebMethod]
    public int Subtract(int x, int y)
    {
        return x - y;
    }
    [WebMethod]
    public int Multiply(int x, int y)
    {
        return x * y;
    }
    [WebMethod]
    public int Division(int x, int y)
    {
        return x / y;
    }
```

Note that `[WebMethod]` tag allows the methods to be accessible to external clients. Now replace the code:

Collapse

```
[WebService(Namespace = "http://tempuri.org/")]
```

with:

Collapse

```
[WebService(Namespace="http://tempuri.org/",
Description="A Simple Web Calculator Service",
Name="CalculatorWebService")]
```

The *Description* attribute gives external clients a brief description of the service; the *Name* attribute lets external clients refer to the service as `CalculatorWebService` rather than `Service`.

Installing the Web Service in IIS

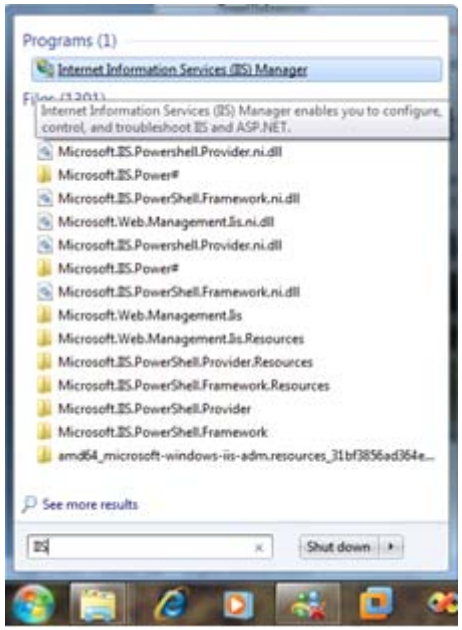
This project uses IIS Server 7.0 running on a Windows 7 PC.

Activate IIS server by:

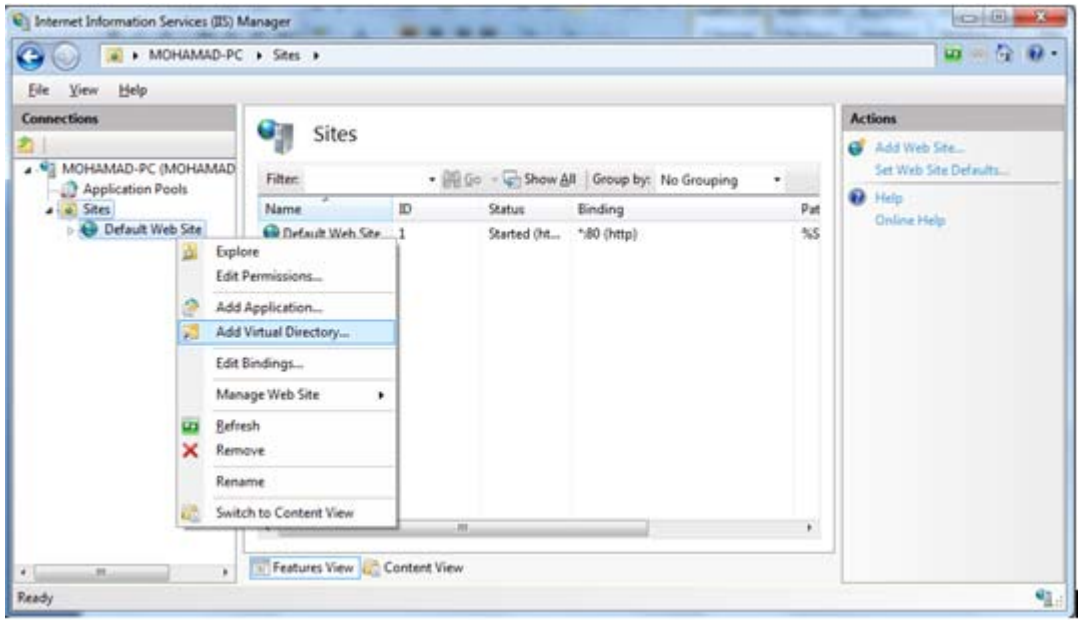
Start Menu / typing IIS in the search bar / Internet Information Services (IIS) Manager

or by: Control Panel / Administrative Tools / Internet Information Services (IIS) Manager

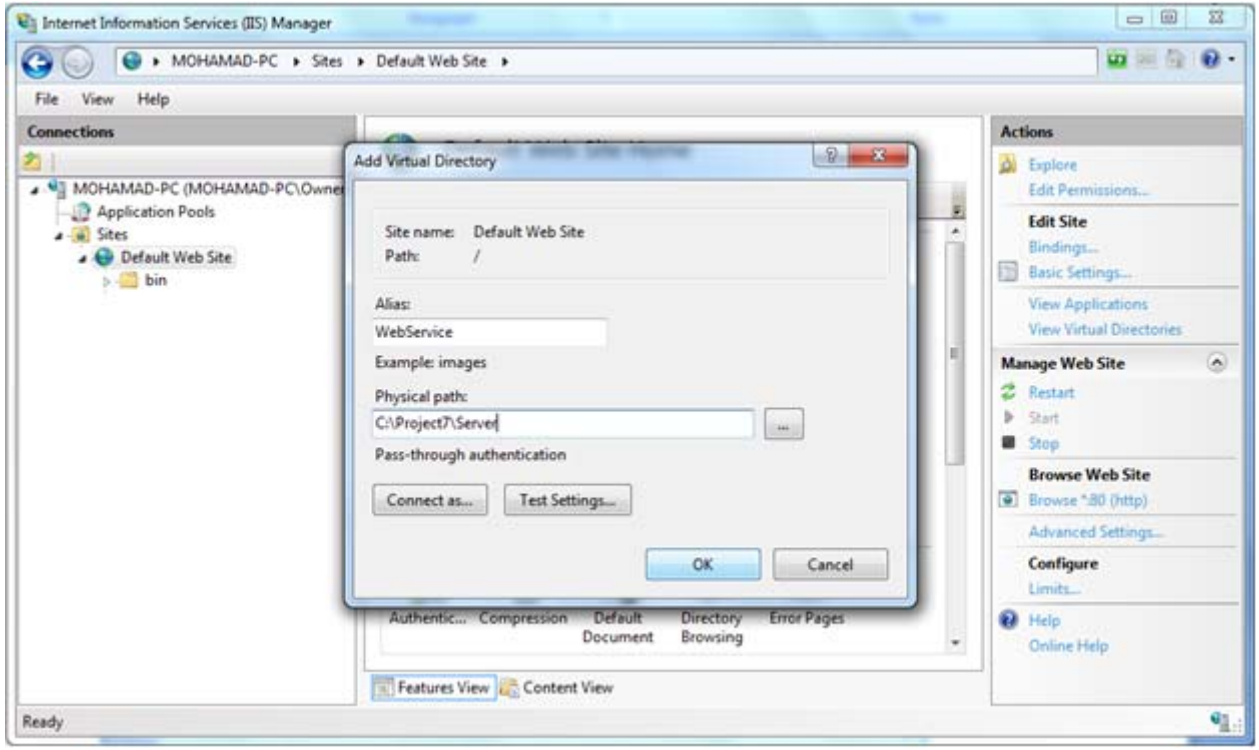
If you couldn't find IIS, then probably it's not installed, so consult Appendix A to know how you can activate this program under Windows 7.



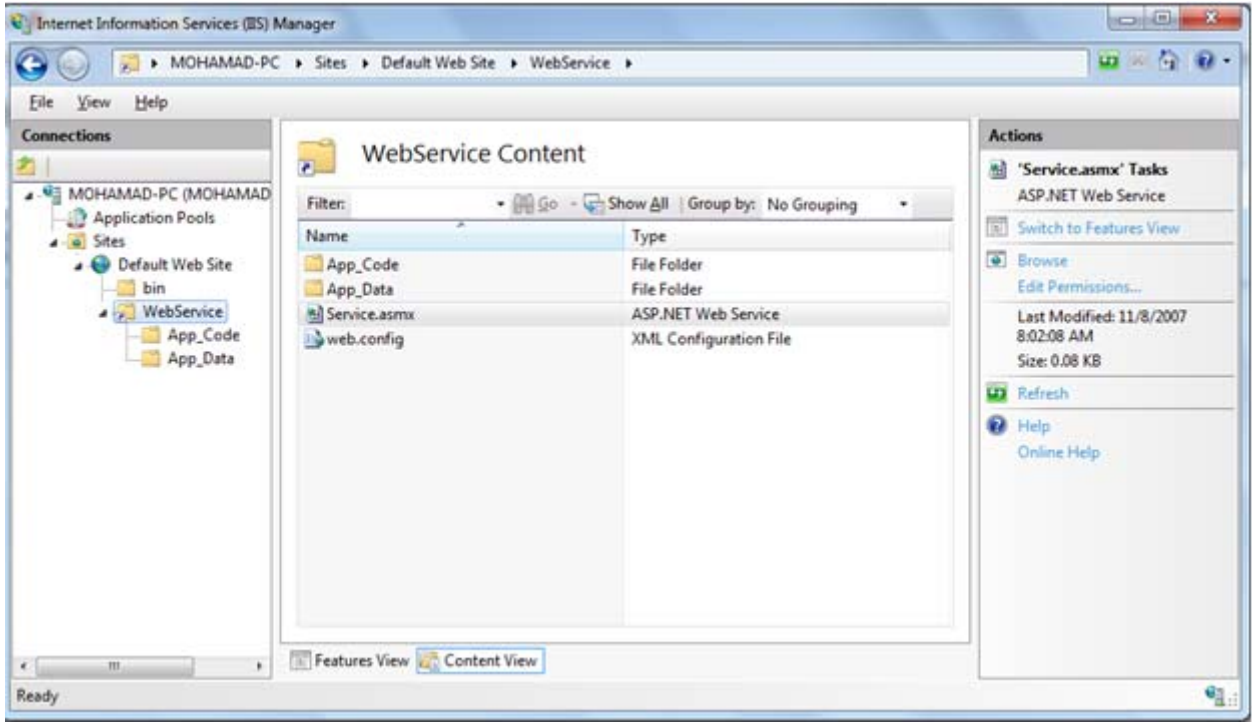
Expand the tree *Sites*, then right click on *Default Web Site*, and choose *Add Virtual Directory*.



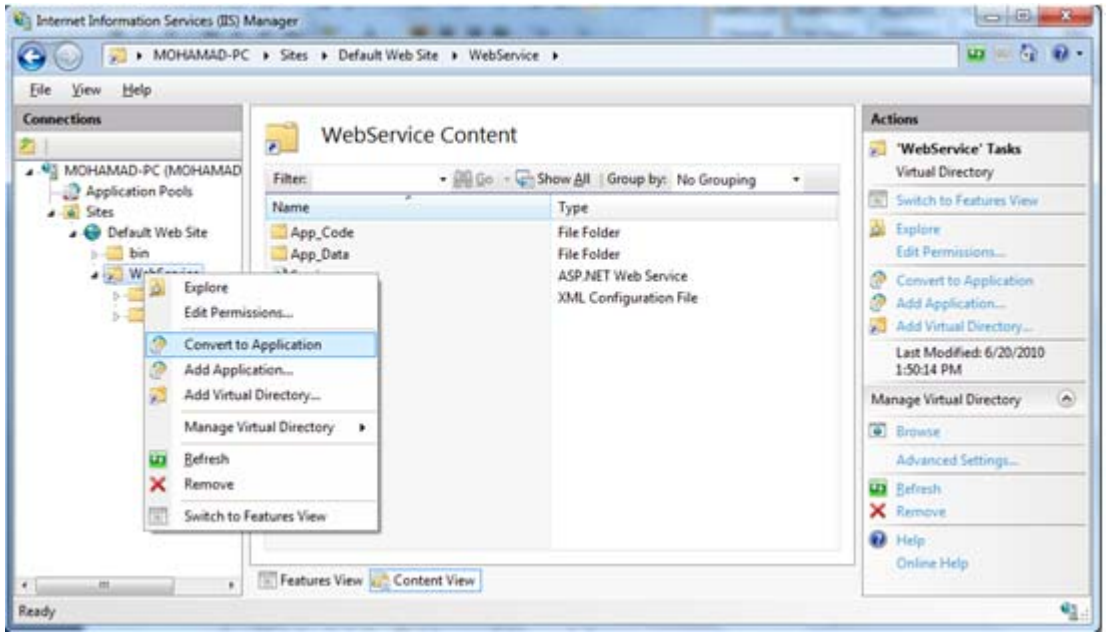
Enter *WebService* in the Alias field, and *C:\Project7\Server* in the *Physical path* field.



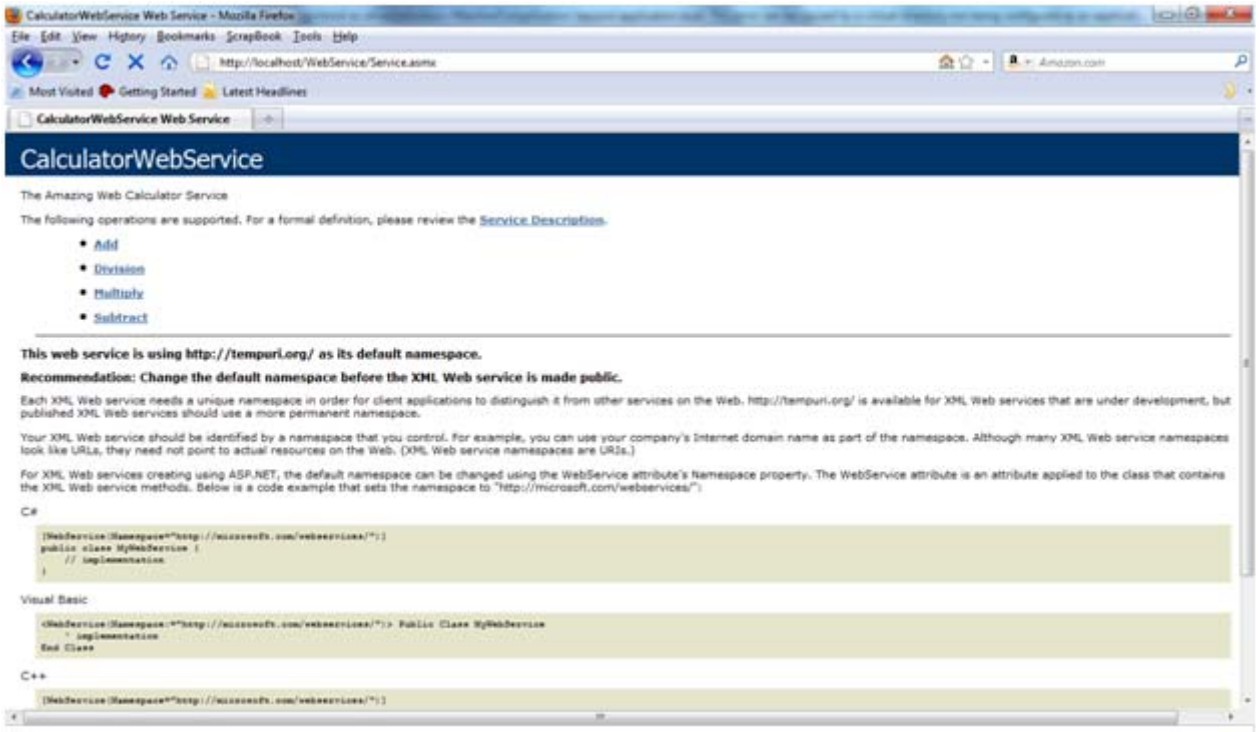
Click on the *WebService* folder and then switch IIS to *Content View* in order to see the *Service.asmx* and the *web.config* files.



Since we've manually created the Virtual Directory *WebService* without the help of Visual Studio testing mode, you should right click the *WebService* folder and choose *Convert to Application* followed by clicking OK.



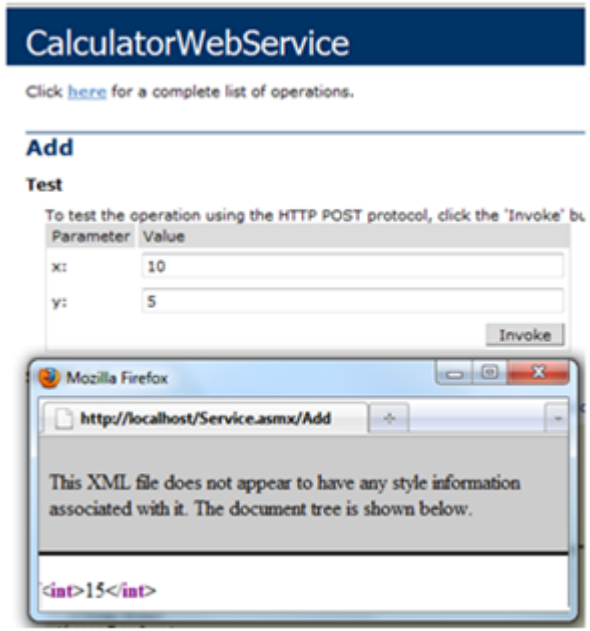
Now open your browser and goto *http://localhost/WebService/Service.asmx*, you should see a screen similar to this:

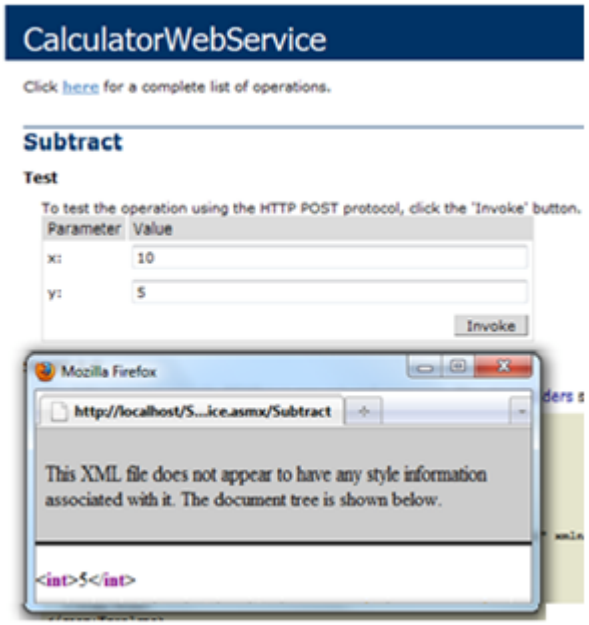


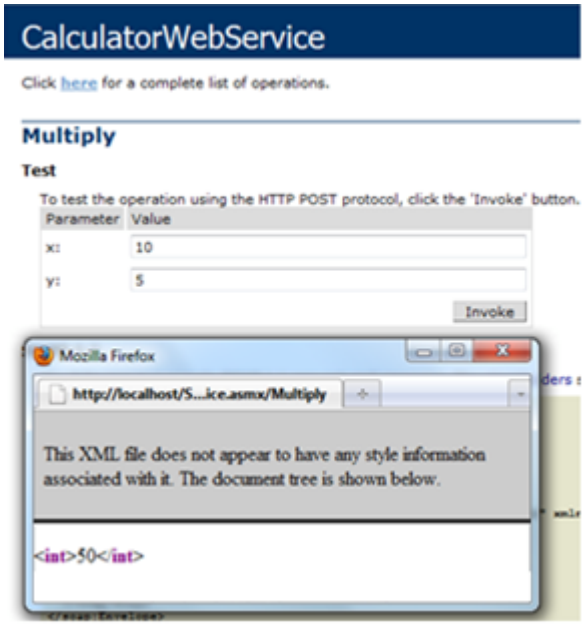
Note: If you didn't see this screen, then probably IIS server isn't configured properly, so consult Appendix B to learn how to configure IIS for the first time.

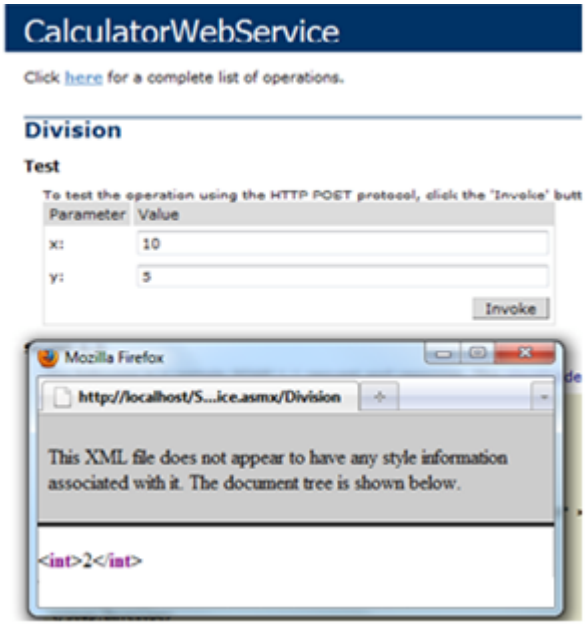
Testing the Web Service on the Server Side

On the *CalculatorWebService* page, you could see four links to the `Add`, `Division`, `Multiply`, and `Subtract` methods that we've implemented in C#; clicking on any of those links will take you to a new page where you could test your methods.





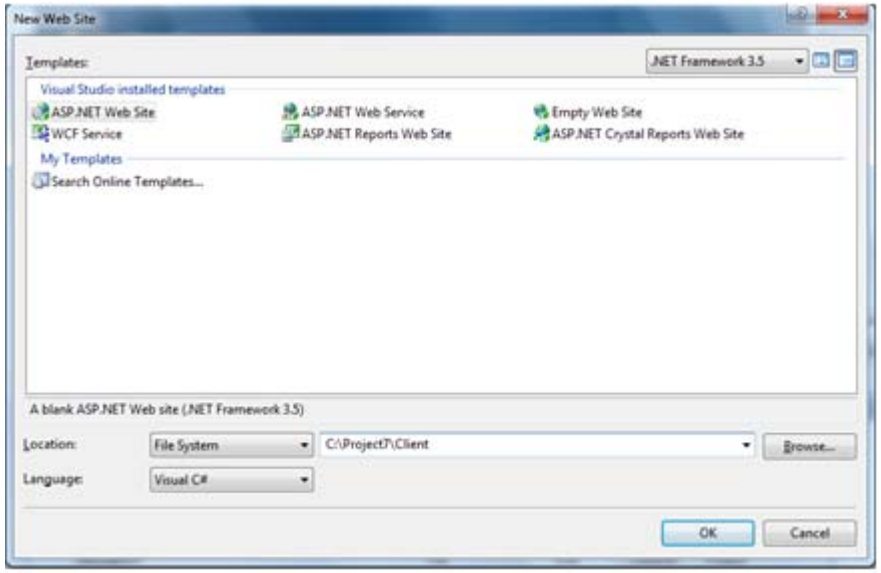




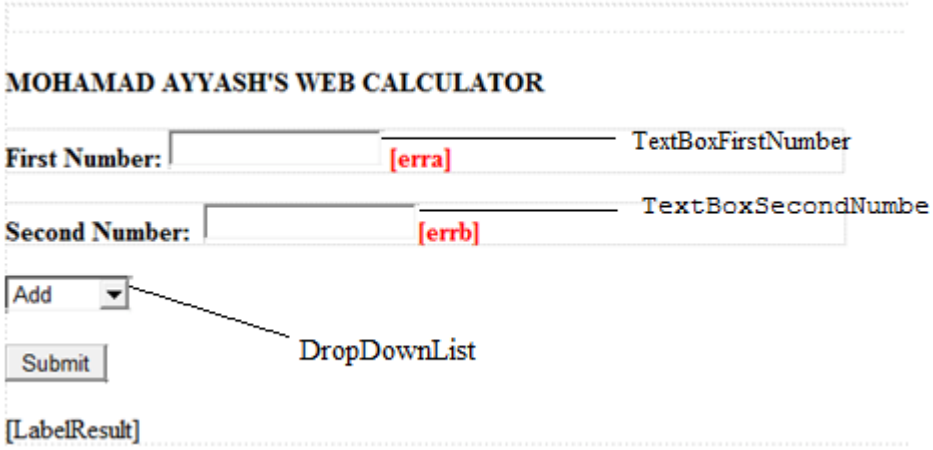
Creating the ASP.NET Web Client

Launch Visual Studio, and then go to File / New / Web Site...

Choose *ASP.NET Web Site* as the template and name your project *Client*.



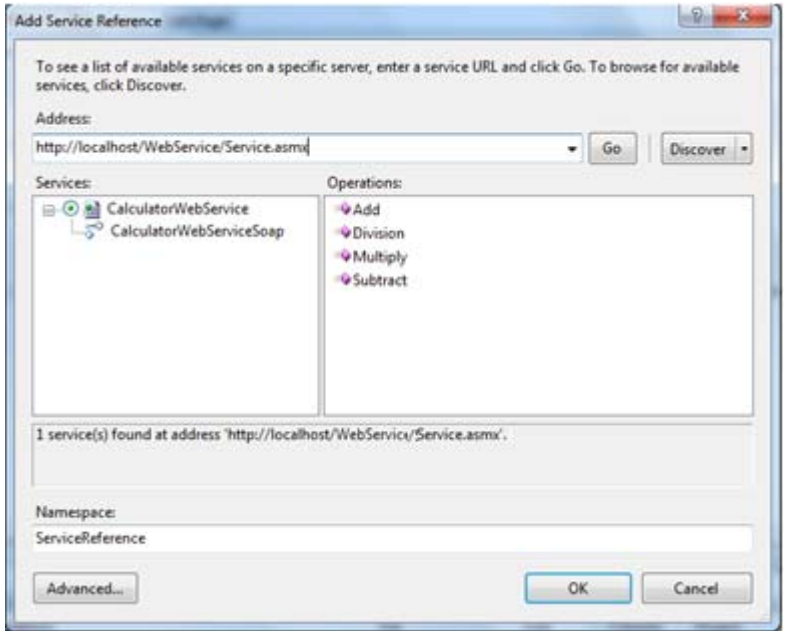
Create an interface of your ASP.NET page similar to this:



Now let's add our web service as a service reference to our project as follows:

Right Click the Project in the solution explorer / choose *Add Service Reference* / enter *http://localhost/WebService /Service.asmx* in the address field /click *Go* to view the imported functions / choose *ServiceReference* as your

namespace / click OK.



Edit your *Default.aspx.cs* source to add the method `GetResult` that takes as an input two number strings and an integer function which corresponds to the four basic calculator operations we need.

Collapse

```
private string GetResult(string firstNumber, string secondNumber, int function)
    {
        ServiceReference.CalculatorWebServiceSoapClient client =
            new ServiceReference.CalculatorWebServiceSoapClient();
        int a, b;
        string result = null;

        erra.Text = "";
        errb.Text = "";

        if (!int.TryParse(firstNumber, out a))
        {
            erra.Text = "Must be a valid 32-bit integer!";
            return "";
        }

        if (!int.TryParse(secondNumber, out b))
        {
            errb.Text = "Must be a valid 32-bit integer!";
            return "";
        }

        try
        {
            switch (function)
            {
                case 0:
                    result = firstNumber + " + " + secondNumber +
                        " = " + client.Add(a, b);
                    break;
                case 1:
                    result = firstNumber + " - " + secondNumber + " = "
                    + client.Subtract(a, b);
                    break;
                case 2:
                    result = firstNumber + " * " + secondNumber + " =
                        " + client.Multiply(a, b);
                    break;
                case 3:
                    result = firstNumber + " / " +
                    secondNumber + " = " + client.Division(a, b);
                    break;
            }
        }
        catch (Exception e)
        {
            LabelResult.ForeColor = System.Drawing.Color.Red;
            result = "Cannot Divide by Zero!";
        }
        return result;
    }
```

Note the statement:

Collapse

```
ServiceReference.CalculatorWebServiceSoapClient client =
            new ServiceReference.CalculatorWebServiceSoapClient();
```

which allows the client object to access the web service methods. `ServiceReference` is the namespace of the web service you chose earlier. `CalculatorWebServiceSoapClient` establishes the SOAP connection with client (i.e. sends requests and receives responses in the form of SOAP XML messages between the proxy server and the proxy client).

Finally, add the Submit Button event handler with the following code to access the `GetResult` method you created earlier.

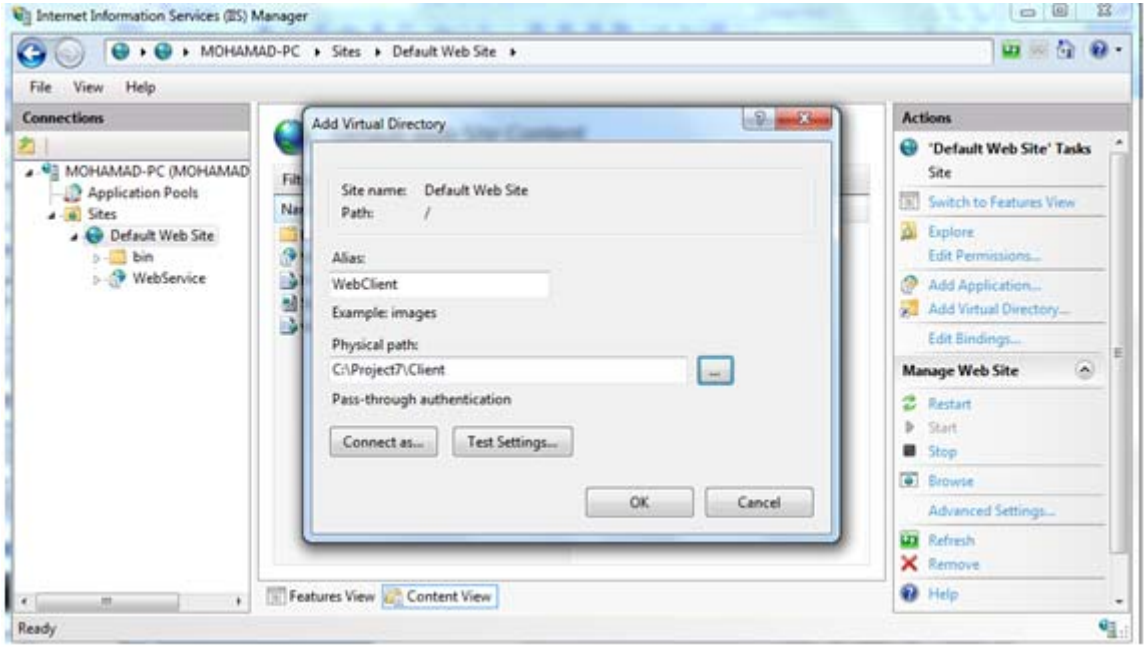
Collapse

```
protected void btnSubmit_Click(object sender, EventArgs e)
    {
        LabelResult.ForeColor = System.Drawing.Color.Black;
        LabelResult.Text = GetResult(TextBoxFirstNumber.Text,
        TextBoxSecondNumber.Text,
        DropDownList.SelectedIndex);
    }
```

Installing the Web Client in IIS Server

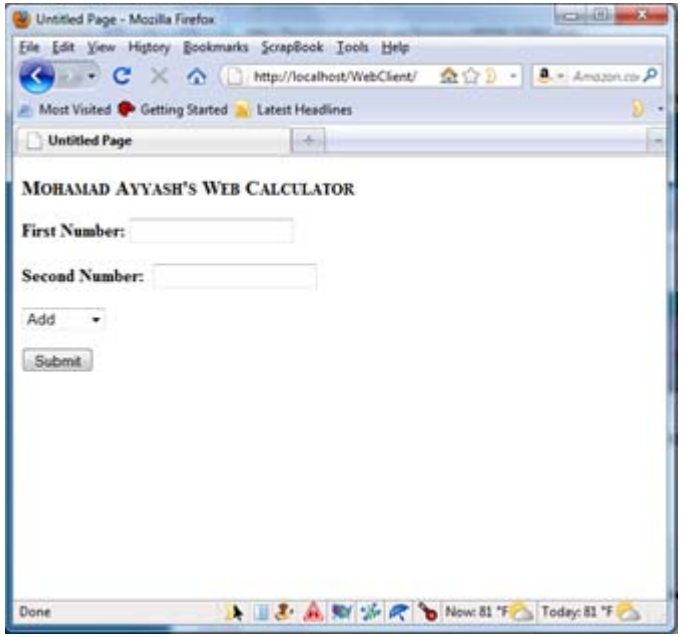
Now you're ready to make use of the web service with a small ASP web page that consumes this service.

Create a new Virtual Directory in IIS and choose *WebClient* as an Alias; and the folder *C:\Project7\Client* as a *Physical Path*.



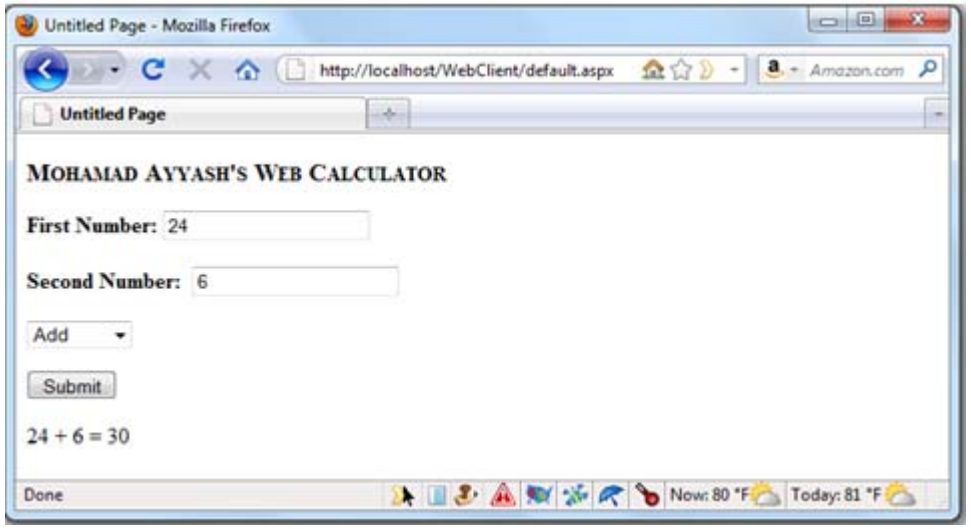
And as you did for the *WebService* Virtual Directory, right click on the *WebClient* directory and choose *Convert to Application* followed by an OK.

Finally go to the browser and type in *http://localhost/WebClient/*, you should see the figure below:



Testing the Web Service on the ASP.NET Client Side

Now that we've successfully installed the client page that makes use of our calculator web service, we still need to validate the client side results.



Installing and Testing the Web Client in a Java 1.6 Class

Regardless of whether we know the programming language that implements the web service or not, we can easily link applications of different platforms. Web services display to their clients standard XML pages called WSDL, which gives them an idea about what functions they implement. For instance, navigating to *http://localhost/WebService/Service.asmx?wsdl* would generate an XML page with special elements. The ability to read through the WSDL document and to locate elements like: **type**, **message**, **portType**, **binding**, **service** is very crucial, as it tells client developers about the functions hosted by the service (not a documentation). For more information about WSDL documents, visit www.w3schools.com/wsdl/wsdl_documents.asp.

Fortunately enough for JAVA 1.6 developers, is the existence of the JAVA *jax-ws* package that generates the proxy classes necessary for communicating with the web server.

First let's build the java project in *C:\Project7\java_proj\webClient*.

So, start by creating a new java class called *cl.java*. (Make sure your current directory is *C:\Project7\java_proj\webClient*).

Now let's run the *jax-ws* utility by typing the following in the command line:

Collapse

```
C:\Project7\java_proj\webClient >wsimport -keep -p webPack.serv
http://localhost/Service.asmx?wsdl
```

The *wsimport* command will generate the proxy classes we need, *-keep* keeps the generated source files, and *webPack.serv* is the package containing the proxy classes. You could see by examining the **service** section of the WSDL file and by browsing through the *webPack\serv* folders that you have the two main classes `CalculatorWebService` and `CalculatorWebServiceSoap`.

Next, let's edit our *cl.java* client as follows:

Collapse

```
import webPack.serv.*;

public class cl
{
        public static void main(String[] args)
        {
                CalculatorWebService service = new CalculatorWebService ();
CalculatorWebServiceSoap soap = service.getPort(CalculatorWebServiceSoap.class);
                int x = 5,y = 10;
                System.out.println(soap.Add(x,y));

        }
}
```

The `getPort` method returns the 'stub' of the `CalculatorWebService` class as defined by the web service and lets you access the four calculator methods defined. For instance, the *soap.Add(int,int)* method actually calls the server to perform the *Add* function defined there, and then directs the return value to the client.

Testing this class is easy; just compile as follows:

Collapse

```
>javac –classpath . cl.java
```

Then run this program as follows:

Collapse

```
> java cl
```

You'll get "15" as an output.

Installing and Testing the Web Client in PHP

PHP allows the easy creation and consumption of SOAP based web services through the NuSOAP toolkit. To learn how to download and configure this API toolkit, visit http://sourceforge.net/projects/nusoap/.
First, you need to include the *nusoap* library so type in your PHP shell:

Collapse

```
require_once('nusoap.php');
```

Then you need to connect to our web service's WSDL in order to generate the SOAP client, so type in:

Collapse

```
$wsdl="http://localhost/WebService/Service.asmx?wsdl";
$client=new soapclient($wsdl, 'wsdl');
```

Now that the SOAP `$client` is ready, we could then simply type in:

Collapse

```
$param=array(
'x'=>10,
'y'=>5
);
echo $client->call('add', $param);
```

And we'll get '15' as expected.

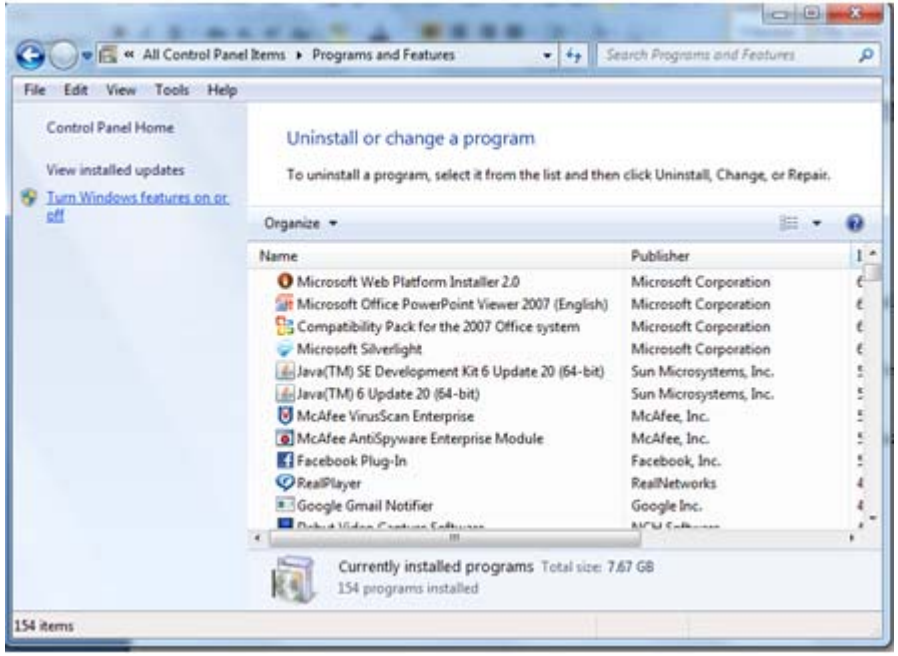
Actually the way to access web services in PHP/NuSOAP is very similar to that in the .NET Framework.

Appendix A: Activating the IIS server and ASP.NET under Windows 7

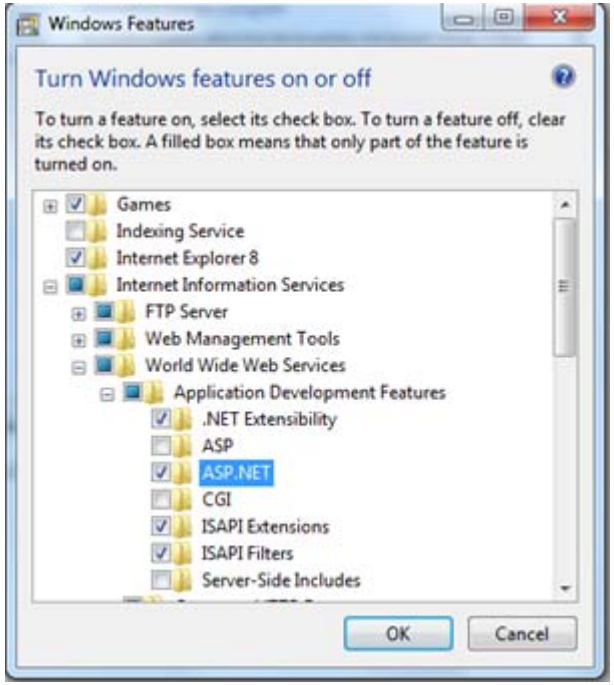
In most Windows computers, IIS server and ASP.NET are not installed by default as they're an optional Windows component; however, the installation process runs smoothly as follows.

Goto *Control Panel* and choose *Programs and Features* (you may need to click View by: Large Icons in order to see the Programs and Features icon).

On the left margin of the Programs and Features window, choose *Turn Windows Features on or off* you may need administrative privileges in order to proceed.



Next, expand the *Internet Information Services* tree to ensure the installation of IIS server. Then expand the World Wide Web Services tree and then expand the *Application Development Features* and make sure that the *ASP.NET* node icon is checked.

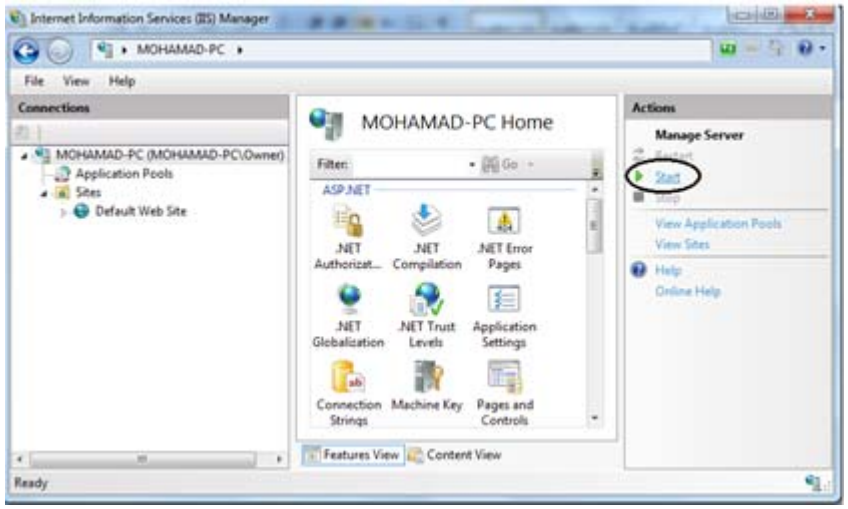


Finally click OK to install the IIS and ASP.NET server.

Appendix B: Configuring the IIS Server for First Time Usage

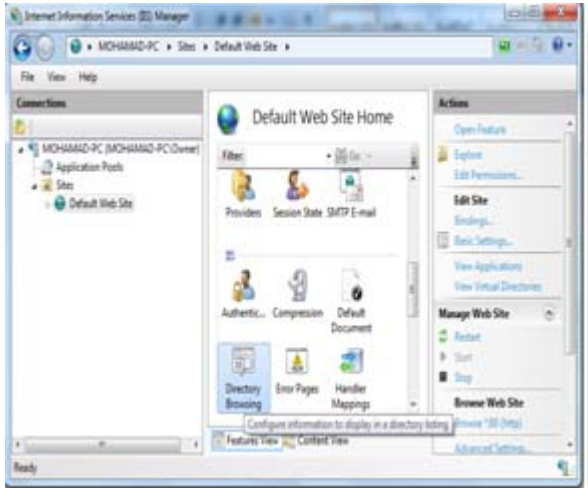
To launch IIS server under Windows 7, go to the start menu and type IIS - you should see *Internet Information Services (IIS) Manager*.

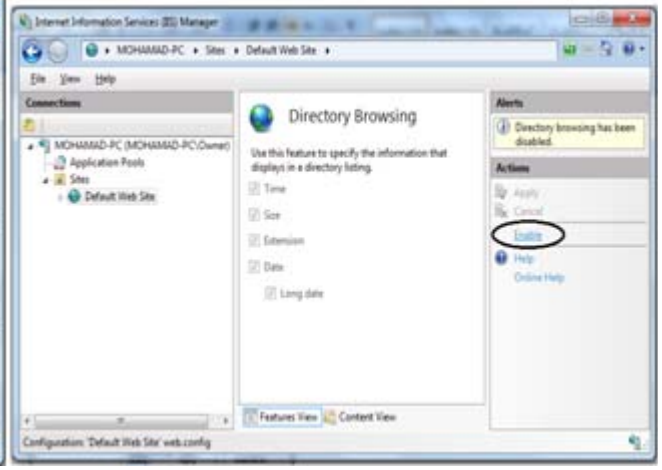
First, you should start the IIS server, so click Start on the right margin labelled *Action*.



Now that the IIS server has started, you may (optionally) want to enable directory listing in order to permit users to browse the contents of your web server.

To do that, click on the *Default Web Site* tree in the left margin named Connections and then double click on the *Directory Browsing* icon located in the middle of the screen. In the *Actions*, right margin click *Enable*.





You're now ready to view your site. Just launch your browser and type http://localhost in your browser.

Conclusion

I hope you now understand the importance of web services and their role in facilitating the concept of distributed

computing between different systems running different programming languages. If you're interested in using some freely available web services in your applications, visit www.webservicex.net/WS/wscatlist.aspx for more information.

License

About the Author

Mohamad K. Ayyash

Tester / Quality Assurance
ford.com
United States

Member

A Computer Engineering student at the University of Michigan - Dearborn and a SYNC QC/QC employee at Ford Motor Company.

Article Top

Comments and Discussions

My vote of 5	**Yusuf**	**18:17 18 Jan '11**
Re: My vote of 5	Mohamad K. Ayyash	18:48 5 Feb '11
my 5	**Omar Gamil**	**6:10 14 Sep '10**
Re: my 5	Mohamad K. Ayyash	7:22 14 Sep '10
Excellent	**Md. Marufuzzaman**	**0:25 11 Aug '10**
Re: Excellent	Mohamad K. Ayyash	8:54 11 Aug '10
Re: Excellent	Md. Marufuzzaman	9:13 11 Aug '10
Re: Excellent [modified]	Mohamad K. Ayyash	9:18 11 Aug '10
My vote of 5	**Rick Benadez**	**13:50 8 Aug '10**
Re: My vote of 5	Mohamad K. Ayyash	15:28 8 Aug '10
Why do i encounter errors trying to browse my web service	**slimtugo**	**11:16 8 Aug '10**
Re: Why do i encounter errors trying to browse my web service	Mohamad K. Ayyash	11:50 8 Aug '10
Re: Why do i encounter errors trying to browse my web service	slimtugo	4:20 9 Aug '10
Re: Why do i encounter errors trying to browse my web service [modified]	Mohamad K. Ayyash	16:46 9 Aug '10
Re: Why do i encounter errors trying to browse my web service	slimtugo	5:04 10 Aug '10
Re: Why do i encounter errors trying to browse my web service	Mohamad K. Ayyash	6:51 10 Aug '10
over all good	**Pranay Rana**	**22:35 21 Jul '10**
My vote of 5	**defwebserver**	**6:22 20 Jul '10**
Re: My vote of 5	Mohamad K. Ayyash	7:02 20 Jul '10
;)	**Dinkas**	**9:22 15 Jul '10**

Last Visit: 5:51 10 Jul '11 Last Update: 10:24 17 Jul '11 **1**

General News Suggestion Question Bug Answer Joke Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+PgUp/PgDown to switch pages.