**WEB SERVICES**

**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.

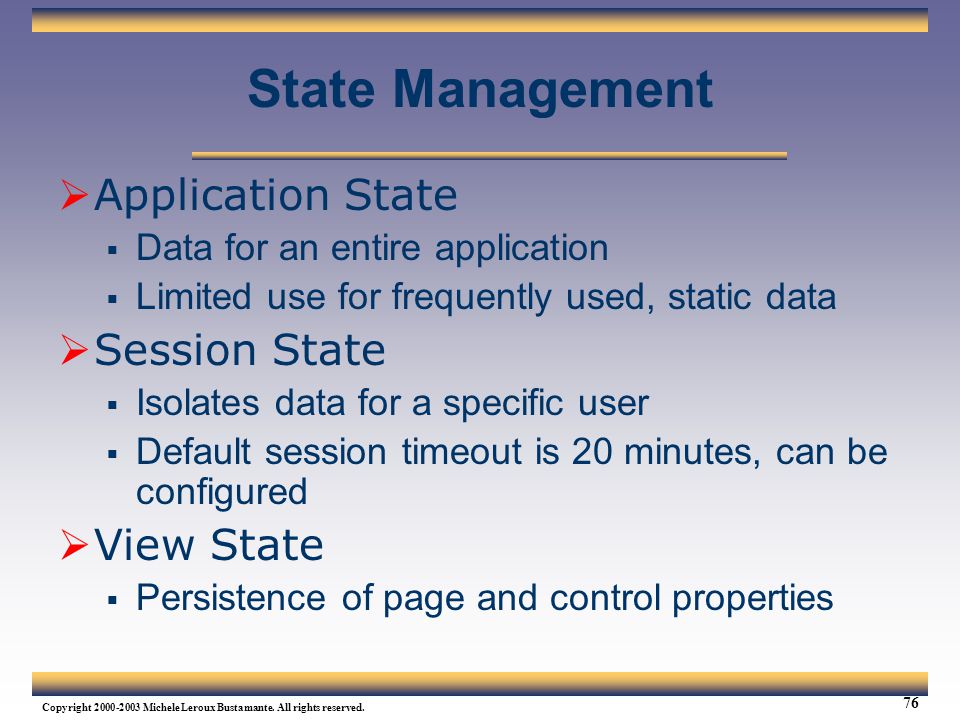
* **Web Portal-**A web portal might obtain top news headlines from an Associated press web service.
* **Weather Reporting-** You can use Weather Reporting web service to display weather information in your personal website.
* **Stock Quote-** You can display latest update of Share market with Stock Quote 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.

**State management**

In traditional Web programming, all information associated with the page and the controls on the page would be lost with each round trip. For example, if a user enters information into a text box, the information would be lost in the round trip from the Browser or client device to the Server. Similarly if we want to store some internal information, data inside the Application will also be lost in each round trip or when we refresh the page.

To overcome this limitation of the traditional Web programming, ASP.NET includes several options that help you preserve data on both a client side as well as server side.  
  
These techniques are commonly known as state Management.  
  
As per my understanding "*State management is nothing but a temporary memory storage to store state of a page or information of a page against various Postback*".

In ASP.NET, there are two types of state management techniques: Client side and Server side but in Web Service, we can have very limited state management techniques. Here in Web Services, we have only Server side State Management.



We have mainly three options of state management in the Web Services, which are shown below.

* Session
* Application
* View state

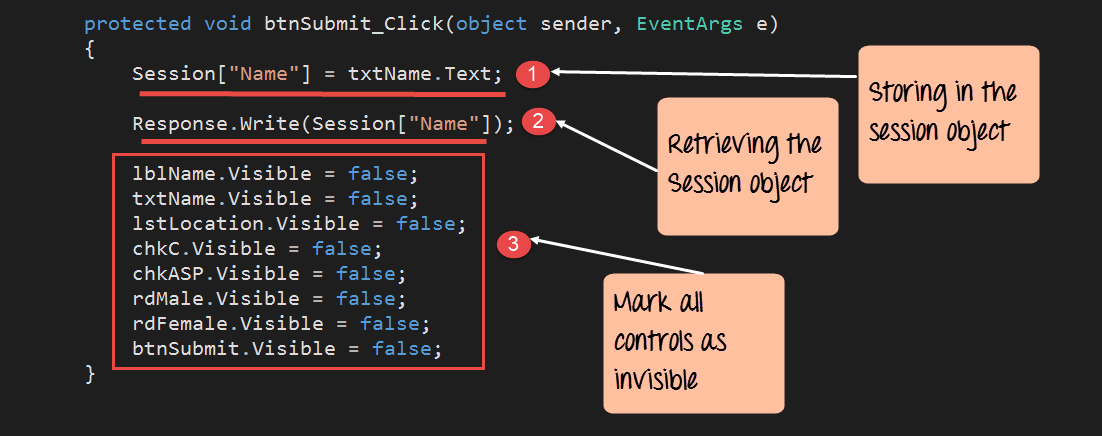
Session State Management

The Web is stateless, which means a new instance of the Web page class is re-created, each time the page is posted to the Server. As we all know, HTTP is a stateless protocol, it can't hold the client information on a page. If the user inserts some information and moves to the next page or clicks any button or does any kind of Postback, the data will be lost and the user will not be able to retrieve the information. What do we need here? We need to store the information.

Hence, Session provides a facility to store the information on the Server memory. It can support any type of object to store along with our own custom objects. For every client, Session data is stored separately, which means Session data is stored on a per client basis.  
For each session we have a unique block of memory allocated in the server. Session object is per client, which means that different clients generate different Session objects.

Session management is a very strong technique to maintain state. Generally session is used to store user's information and/or uniquely identify a user (or say browser). The server maintains the state of user information by using a session ID. When users make a request without a session ID, ASP.NET creates a session ID and sends it with every request and response to the same user.

To use ASP.NET Session object in a Web Service, there are two things, which we need to do which are shown below.

1. [](https://www.guru99.com/images/asp-net/061516_0807_ASPNetIntro43.png)The WebService class should inherit from System.Web.Services.WebService class.
2. The EnableSession property of the WebMethod attribute should be set to true.

we are going to use the Session object to store the name entered in the name textbox field in the page. We are then going to retrieve that value and display it on the page accordingly. Let's add the below code to the Demo.aspx.cs file.

protected void btnSubmit\_Click(object sender,EventArgs e)

{

Session["Name"] = txtName.Text;

Response.Write(Session["Name"]);

lblName.Visible = false;

txtName.Visible = false;

1stLocation.Visible = false;

chkC.Visible = false;

chkASP.Visible = false;

rdMale.Visible = false;

rdFemale.Visible = false;

btnSubmit.Visible = false;

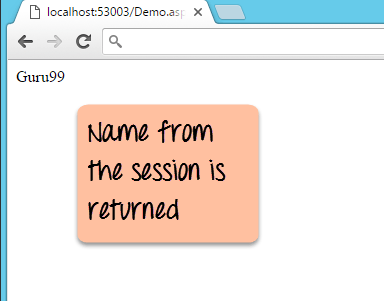
}

**Code Explanation:-**

1. The first line of code takes the value of the Name textbox control and stores it in the Session object. By specifying the code of Session["Name"] , we are giving the property a name called "Name." By specifying a name for the property, it becomes easier to retrieve it at a later point in time.
2. The next line of code retrieves the stored value from the Session object. It then writes this value via the 'Response.Write' method back to the client.
3. Finally, we make all the controls on the form as invisible. If we don't do this, all the controls plus our response values will be displayed together.

Once you make the above changes, you will see the following output

**Output:**

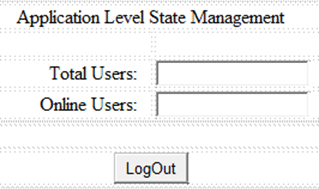
[](https://www.guru99.com/images/asp-net/061516_0807_ASPNetIntro44.png)

From the output, you can see that the Session value of name was retrieved and displayed in the browser.

Application Level State Management

Application Level State Management is used to maintain the state of all the users accessing the webforms present within the website.  
  
The value assigned for an application is considered as an object.  
  
Application object will not have any default expiration period.  
  
Whenever the webserver has been restarted or stopped then the information maintained by the application object will be lost.  
  
If any data is stored on the application object then that information will be shared upon all the users accessing the webserver.

**A Sample program on Application Level State Management:**

* **Design:**   
    
    
    
  Design the form as shown above with two textboxes and one button.
* Right click on application in solution explorer-> Add new item-> select 'GlobalApplicationClass' file and write the following code in it.

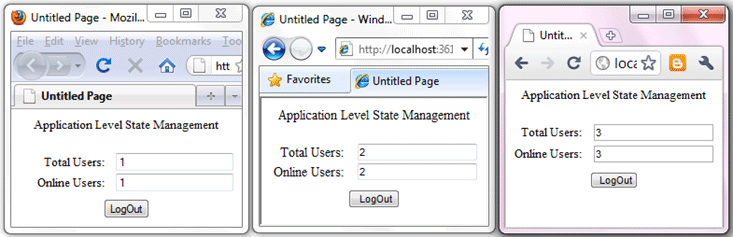
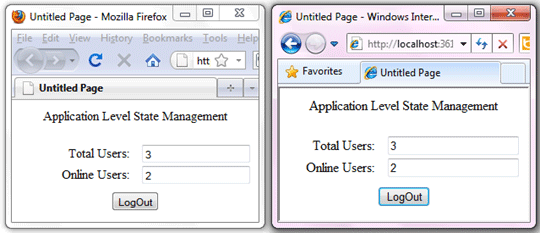
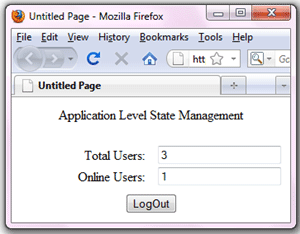
**Global.asax code:**   
  
<%@ Application Language="C#" %>

<script runat="server">

    void Application\_Start(object sender, EventArgs e)   
    {  
        // Code that runs on application startup  
        Application.Lock();  
        Application["tusers"] = 0;  
        Application["ousers"] = 0;  
        Application.UnLock();  
    }  
    void Application\_End(object sender, EventArgs e)   
    {  
        //  Code that runs on application shutdown  
  
    }  
  
    void Application\_Error(object sender, EventArgs e)   
    {   
        // Code that runs when an unhandled error occurs  
    }

    void Session\_Start(object sender, EventArgs e)   
    {  
        // Code that runs when a new session is started  
        int tusers, ousers;  
        Application.Lock();  
        tusers = int.Parse(Application["tusers"].ToString())+1;  
        ousers = int.Parse(Application["ousers"].ToString())+1;  
        Application["tusers"] = tusers;  
        Application["ousers"] = ousers;  
        Application.UnLock();  
    }

    void Session\_End(object sender, EventArgs e)   
    {  
        // Code that runs when a session ends.   
        // Note: The Session\_End event is raised only when the sessionstate mode  
        // is set to InProc in the Web.config file. If session mode is set to StateServer   
        // or SQLServer, the event is not raised.  
        int ousers;  
        Application.Lock();         
        ousers = int.Parse(Application["ousers"].ToString()) - 1;  
        Application["ousers"] = ousers;  
        Application.UnLock();  
    }  
  
</script>  
  
**Code:**  
using System;

public partial class \_Default : System.Web.UI.Page   
{  
    protected void Page\_Load(object sender, EventArgs e)  
    {  
        string tusers, ousers;  
        Application.Lock();  
        tusers = Application["tusers"].ToString();  
        ousers=Application["ousers"].ToString();  
        Application.UnLock();  
        TextBox1.Text = tusers;  
        TextBox2.Text = ousers;  
    }  
    protected void BtnLogOut\_Click(object sender, EventArgs e)  
    {  
        Session.Abandon();   
    }  
}  
  
**Output:**  
  
Execute the application, copy the url and open the same url in other browsers. The result will be as shown below:  
  
  
  
Now, logout and close one browser and refresh the remaining browsers. The result will be as shown below:  
  
  
  
Now, logout and close another browser and resfresh the remaining browser. The result will be as shown below:  
  


1

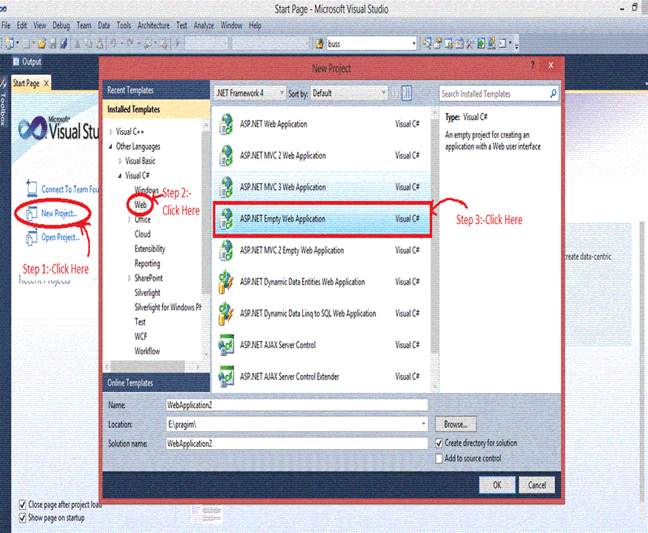
## **View State**

View State is the method to preserve the Value of the Page and Controls between round trips. It is a Page-Level State Management technique. View State is turned on by default and normally serializes the data in every control on the page regardless of whether it is actually used during a post-back.

Now I am showing you an example of what the problem is when we don't use view state.

**Step 1**

Open Visual Studio 2010.

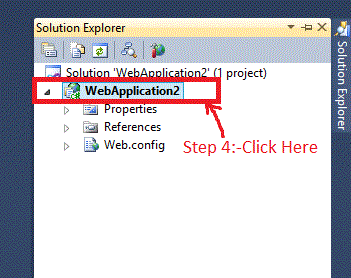


**Step 2**

Then click on "New Project" > "Web" >"ASP.NET Empty Web Application".

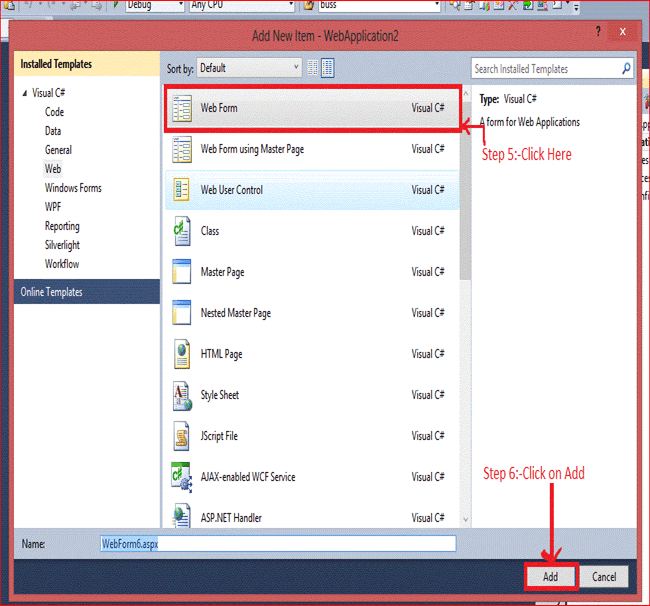
**Step 3**

Now click on Solution Explorer.



**Step 4**

Now right-click on the "ADD" > "New Item" > "Web Form" and **add** the name of the Web Form just like I did in WebForm6.aspx.



**Step 5**

After adding the WebForm6.aspx you will see the following code:

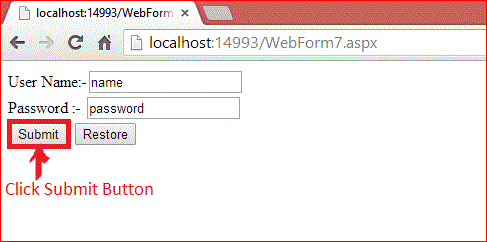
1. <%@ page language="C#" autoeventwireup="true" codebehind="WebForm6.aspx.cs" inherits="view\_state.WebForm6" %>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4. <html xmlns="http://www.w3.org/1999/xhtml">
5. <head runat="server">
6. <title></title>
7. </head>
8. <body>
9. <form id="form1" runat="server">
10. <div>
11. User Name:-<asp:textbox id="TextBox1" runat="server"></asp:textbox>
12. <br />
13. Password  :-<asp:textbox id="TextBox2" runat="server"></asp:textbox>
14. <br />
15. <asp:button id="Button1" runat="server" onclick="Button1\_Click" text="Submit" />
16. <asp:button id="Button3" runat="server" onclick="Button3\_Click" text="Restore" />
17. </div>
18. </form>
19. </body>
20. </html>

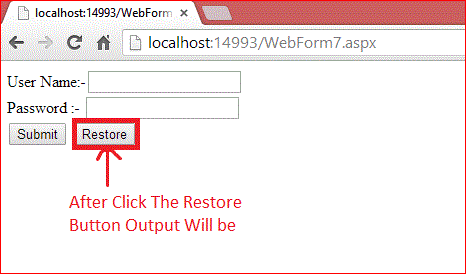
Now write the code as in the following:

1. //Declaration of a and b
2. **public** string a, b;
3. **protected** **void** Button1\_Click(object sender, EventArgs e)
4. {
5. //TextBox1 and TextBox2 Value is Assigning on the variable a and b
6. a = TextBox1.Text;
7. b = TextBox2.Text;
8. //after clicking on Button TextBox value Will be Cleared
9. TextBox1.Text = TextBox2.Text = string.Empty;
10. }
12. **protected** **void** Button3\_Click(object sender, EventArgs e)
13. {
14. //value of variable a and b is assingning on TextBox1 and Textbox2
15. TextBox1.Text = a;
16. TextBox2.Text = b;
17. }

**Output**

Now the output is:





It only happens because all the controls are classes and on the server all the Control Objects are created and then after the round trip the Page is returned to the client's browser in HTML format and the objects are destroyed at the server.

After the Submit button is clicked the value of user name and password is submitted to the server. We cannot restore the value again because after the postback the instance of the control is destroyed and on clicking of the Restore Button the server takes a new request and the server cannot restore the value of the TextBox.

## **Features Of View State**

These are the main features of view state:

1. Retains the value of the Control after post-back without using a session.
2. Stores the value of Pages and Control Properties defined in the page.
3. Creates a custom View State Provider that lets you store View State Information in a SQL Server Database or in another data store.

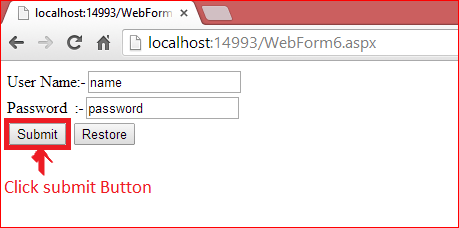
And now I am explaining the stored value in the View State and the remaining steps are the same as the previous.

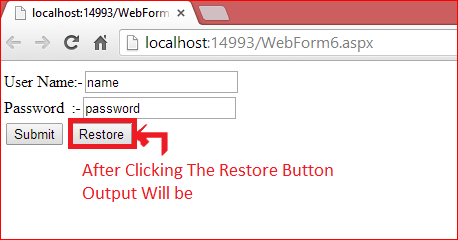
Now write this code,

1. **protected** **void** Button1\_Click(object sender, EventArgs e)
2. {
3. //Value of Textbox1 and TectBox2 is assigin on the ViewState
4. ViewState["name"] = TextBox1.Text;
5. ViewState["password"] = TextBox2.Text;
6. //after clicking on Button TextBox value Will be Cleared
7. TextBox1.Text = TextBox2.Text = string.Empty;
8. }
9. **protected** **void** Button3\_Click(object sender, EventArgs e)
10. {
11. //If ViewState Value is not Null then Value of View State is Assign to TextBox
12. **if** (ViewState["name"] != **null**)
13. {
14. TextBox1.Text = ViewState["name"].ToString();
15. }
16. **if** (ViewState["password"] != **null**)
17. {
18. TextBox2.Text = ViewState["password"].ToString();
19. }
20. }

**Output**

Now the output is,





After clicking on the Submit Button the value of user name and password is submitted in View State and the View State stores the value of user name and password during post-back.

After click on the Restore Button we can get the value again. The Value must be retained during post-back and the values are stored into a base 64 encoded string and this information is then put into the View State Hidden Field.

## **Data Objects That Can be Stored in View state**

1. String
2. Boolean Value
3. Array Object
4. Array List Object
5. Hash Table
6. Custom type Converters

## SOAP Introduction

In today's world, there is huge number of applications which are built on different programming languages. For example, there could be a web application designed in Java, another in .Net and another in PHP.

Exchanging data between applications is crucial in today's networked world. But data exchange between these heterogeneous applications would be complex. So will be the complexity of the code to accomplish this data exchange.

One of the methods used to combat this complexity is to use XML (Extensible Markup Language) as the intermediate language for exchanging data between applications.

Every programming language can understand the XML markup language. Hence, XML was used as the underlying medium for data exchange.

But there are no standard specifications on use of XML across all programming languages for data exchange. That is where SOAP comes in.

SOAP was designed to work with XML over HTTP and have some sort of specification which could be used across all applications. We will look into further details on the SOAP protocol in the subsequent chapters.

## Advantages of SOAP

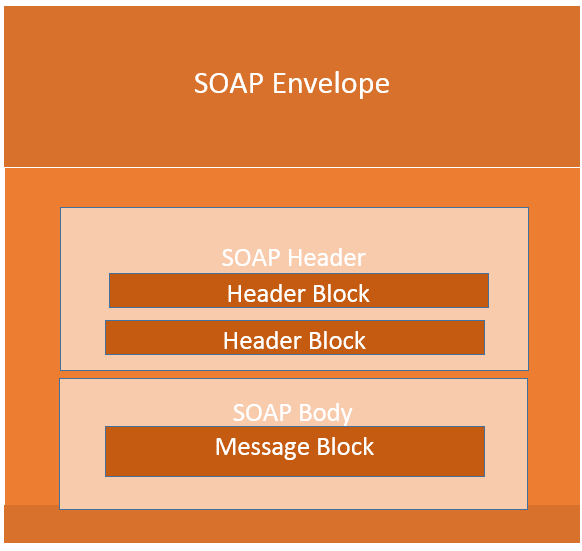
SOAP is the protocol used for data interchange between applications. Below are some of the reasons as to why SOAP is used.

* When developing Web services, you need to have some of language which can be used for web services to talk with client applications. SOAP is the perfect medium which was developed in order to achieve this purpose. This protocol is also recommended by the W3C consortium which is the governing body for all web standards.
* SOAP is a light-weight protocol that is used for data interchange between applications. Note the keyword '**light**.' Since SOAP is based on the XML language, which itself is a light weight data interchange language, hence SOAP as a protocol that also falls in the same category.
* SOAP is designed to be platform independent and is also designed to be operating system independent. So the SOAP protocol can work any programming language based applications on both Windows and[Linux](https://www.guru99.com/unix-linux-tutorial.html)platform.
* It works on the HTTP protocol –SOAP works on the HTTP protocol, which is the default protocol used by all web applications. Hence, there is no sort of customization which is required to run the web services built on the SOAP protocol to work on the World Wide Web.

## SOAP Building blocks

The SOAP specification defines something known as a "**SOAP message**" which is what is sent to the web service and the client application.

The diagram below shows the various building blocks of a SOAP Message.

[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO1.png)

The SOAP message is nothing but a mere XML document which has the below components.

* An Envelope element that identifies the XML document as a SOAP message – This is the containing part of the SOAP message and is used to encapsulate all the details in the SOAP message. This is the root element in the SOAP message.
* A Header element that contains header information – The header element can contain information such as authentication credentials which can be used by the calling application. It can also contain the definition of complex types which could be used in the SOAP message. By default, the SOAP message can contain parameters which could be of simple types such as strings and numbers, but can also be a complex object type.

A simple example of a complex type is shown below.

Suppose we wanted to send a structured data type which had a combination of a "Tutorial Name" and a "Tutorial Description," then we would define the complex type as shown below.

The complex type is defined by the element tag <xsd:complexType>. All of the required elements of the structure along with their respective data types are then defined in the complex type collection.

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Tutorial Name" type="string"/>

<xsd:element name="Tutorial Description" type="string"/>

</xsd:sequence>

</xsd:complexType>

* A Body element that contains call and response information – This element is what contains the actual data which needs to be sent between the web service and the calling application. Below is an example of the SOAP body which actually works on the complex type defined in the header section. Here is the response of the Tutorial Name and Tutorial Description that is sent to the calling application which calls this web service.

<soap:Body>

<GetTutorialInfo>

<TutorialName>Web Services</TutorialName>

<TutorialDescription>All about web services</TutorialDescription>

</GetTutorialInfo>

</soap:Body>

## SOAP Message Structure

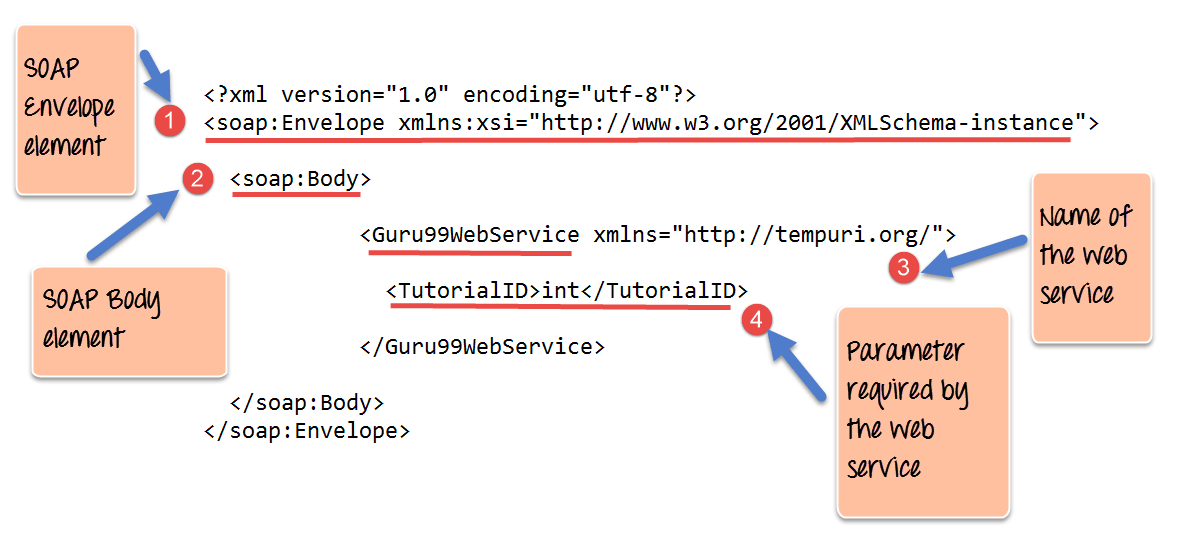
One thing to note is that SOAP messages are normally auto-generated by the web service when it is called.

Whenever a client application calls a method in the web service, the web service will automatically generate a SOAP message which will have the necessary details of the data which will be sent from the web service to the client application.

As discussed in the previous topic, a simple SOAP Message has the following elements –

* The Envelope element
* The header element and
* The body element
* The Fault element (Optional)

Let's look at an example below of a simple SOAP message and see what element actually does.

[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO2.png)

1. As seen from the above SOAP message, the first part of the SOAP message is the envelope element which is used to encapsulate the entire SOAP message.
2. The next element is the SOAP body which contains the details of the actual message.
3. Our message contains a web service which has the name of "Guru99WebService".
4. The "Guru99Webservice" accepts a parameter of the type 'int' and has the name of TutorialID.

Now, the above SOAP message will be passed between the web service and the client application.

You can see how useful the above information is to the client application. The SOAP message tells the client application what is the name of the Web service, and also what parameters it expects and also what is the type of each parameter which is taken by the web service.

## SOAP Envelope Element

The first bit of the building block is the SOAP Envelope.

The SOAP Envelope is used to encapsulate all of the necessary details of the SOAP messages, which are exchanged between the web service and the client application.

The SOAP envelope element is used to indicate the beginning and end of a SOAP message. This enables the client application which calls the web service to know when the SOAP message ends.

The following points can be noted on the SOAP envelope element.

* Every SOAP message needs to have a root Envelope element. It is absolutely mandatory for SOAP message to have an envelope element.
* Every Envelope element needs to have at least one soap body element.
* If an Envelope element contains a header element, it must contain no more than one, and it must appear as the first child of the Envelope, before the body element.
* The envelope changes when SOAP versions change.
* A v1.1-compliant SOAP processor generates a fault upon receiving a message containing the v1.2 envelope namespace.
* A v1.2-compliant SOAP processor generates a Version Mismatch fault if it receives a message that does not include the v1.2 envelope namespace.

Below is an example of version 1.2 of the SOAP envelope element.

<?xml version="1.0"?>

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope" SOAP-ENV:encodingStyle=" http://www.w3.org/2001/12/soap-encoding">

<soap:Body>

<Guru99WebService xmlns="http://tempuri.org/">

<TutorialID>int</TutorialID>

</Guru99WebService>

</soap:Body>

</SOAP-ENV:Envelope>

**The Fault message**

When a request is made to a SOAP web service, the response returned can be of either 2 forms which are a successful response or an error response. When a success is generated, the response from the server will always be a SOAP message. But if SOAP faults are generated, they are returned as "HTTP 500" errors.

The SOAP Fault message consists of the following elements.

1. **<faultCode>**- This is the code that designates the code of the error. The fault code can be either of any below values
   1. SOAP-ENV:VersionMismatch – This is when an invalid namespace for the SOAP Envelope element is encountered.
   2. SOAP-ENV:MustUnderstand - An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood.
   3. SOAP-ENV:Client - The message was incorrectly formed or contained incorrect information.
   4. SOAP-ENV:Server - There was a problem with the server, so the message could not proceed.
2. **<faultString>** - This is the text message which gives a detailed description of the error.
3. **<faultActor>** **(Optional)**- This is a text string which indicates who caused the fault.
4. **<detail>(Optional)** - This is the element for application-specific error messages. So the application could have a specific error message for different business logic scenarios.

**Example for Fault Message**

An example of a fault message is given below. The error is generated if the scenario wherein the client tries to use a method called TutorialID in the class GetTutorial.

The below fault message gets generated in the event that the method does not exist in the defined class.

<?xml version='1.0' encoding='UTF-8'?>

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<SOAP-ENV:Body>

<SOAP-ENV:Fault>

<faultcode xsi:type="xsd:string">SOAP-ENV:Client</faultcode>

<faultstring xsi:type="xsd:string">

Failed to locate method (GetTutorialID) in class (GetTutorial)

</faultstring>

</SOAP-ENV:Fault>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

**Output:**

When you execute the above code, it will show the error like "Failed to locate method (GetTutorialID) in class (GetTutorial)"

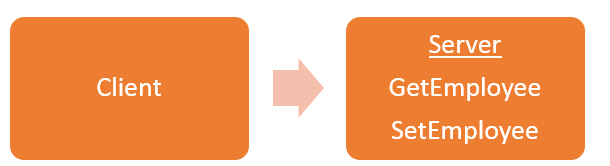
## SOAP Communication Model.

All communication by SOAP is done via the HTTP protocol. Prior to SOAP, a lot of web services used the standard RPC (Remote Procedure Call) style for communication. This was the simplest type of communication, but it had a lot of limitations.

Let's consider the below diagram to see how this communication works. In this example, let's assume the server hosts a web service which provided 2 methods as

* **GetEmployee** - This would get all Employee details
* **SetEmployee** – This would set the value of the details like employees dept, salary, etc. accordingly.

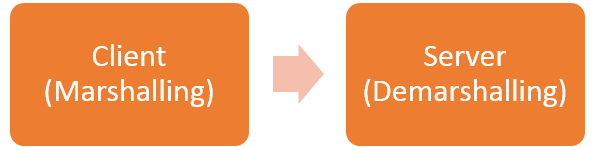
In the normal RPC style communication, the client would just call the methods in its request and send the required parameters to the server, and the server would then send the desired response.

[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO3.png)

The above communication model has the below serious limitations

1. **Not Language Independent** – The server hosting the methods would be in a particular programming language and normally the calls to the server would be in that programming language only.
2. **Not the standard protocol**– When a call is made to the remote procedure, the call is not carried out via the standard protocol. This was an issue since mostly all communication over the web had to be done via the HTTP protocol.
3. **Firewalls**– Since RPC calls do not go via the normal protocol, separate ports need to be open on the server to allow the client to communicate with the server. Normally all firewalls would block this sort of traffic, and a lot of configuration was generally required to ensure that this sort of communication between the client and the server would work.

To overcome all of the limitations cited above, SOAP would then use the below communication model

[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO4.png)

1. The client would format the information regarding the procedure call and any arguments into a SOAP message and sends it to the server as part of an HTTP request. This process of encapsulating the data into a SOAP message was known as **Marshalling.**
2. The server would then unwrap the message sent by the client, see what the client requested for and then send the appropriate response back to the client as a SOAP message. The practice of unwrapping a request sent by the client is known as **Demarshalling.**

## Practical SOAP Example

Let see a practical example,

Probably one of the best ways to see how SOAP messages get generated is to actually see a web service in action.

This topic will look at using the Microsoft.Net framework to build an ASMX web service. This type of web service supports both SOAP version 1.1 and version 1.2.

ASMX web services automatically generate the Web Service Definition Language (WSDL) document. This WSDL document is required by the calling client application so that the application knows what the web service is capable of doing.

In our example, we are going to create a simple web service, which will be used to return a string to the application which calls the web service.

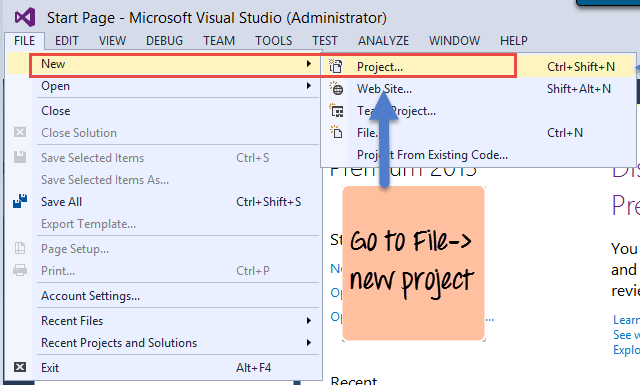
This web service will be hosted in an[Asp.Net](https://www.guru99.com/asp-net-tutorial.html)web application. We will then invoke the web service and see the result that is returned by the web service.

Visual Studio will also show us what the SOAP message being passed between the web service and the calling application.

The first pre-requisite to setup our Web service application which can be done by following the below steps.

Please ensure that you have Visual Studio 2013 installed on your system for this example.

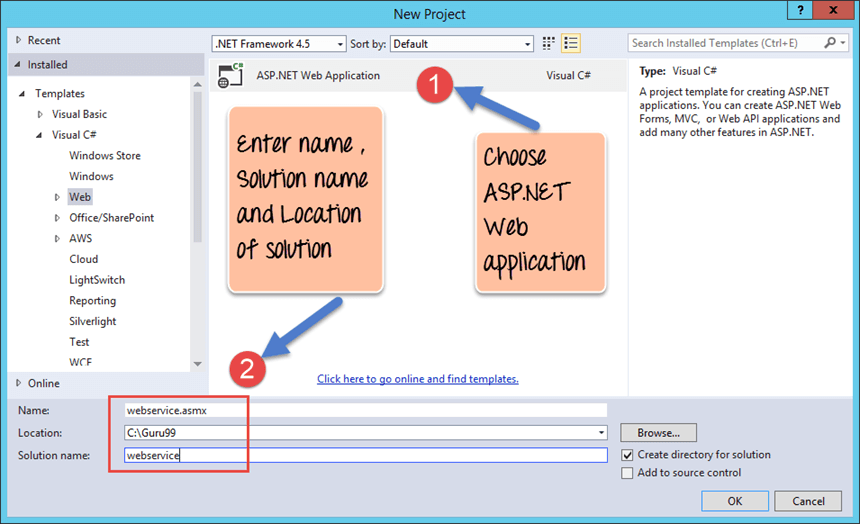
**Step 1)** The first step is to create an empty ASP.Net Web application. From Visual Studio 2013, click on the menu option File->New project.

[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO5.png)

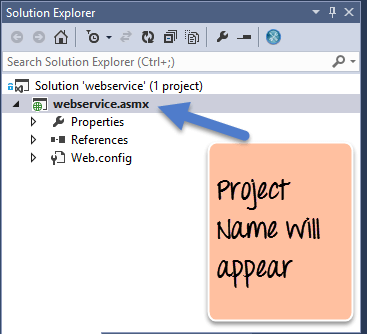
Once you click on the New Project option, Visual Studio will then give you another dialog box for choosing the type of project and to give the necessary details of the project. This is explained in the next step.

**Step 2)** In this step,

1. Ensure to first choose the[C#](https://www.guru99.com/c-tutorial.html)web template of ASP.NET Web application. The project has to be of this type in order to create web services project. By choosing this option, Visual Studio will then carry out the necessary steps to add required files which are required by any web-based application.
2. Give a name for your project which in our case has been given as webservice.asmx. Then ensure to give a location where the project files will be stored.

[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO6.png)

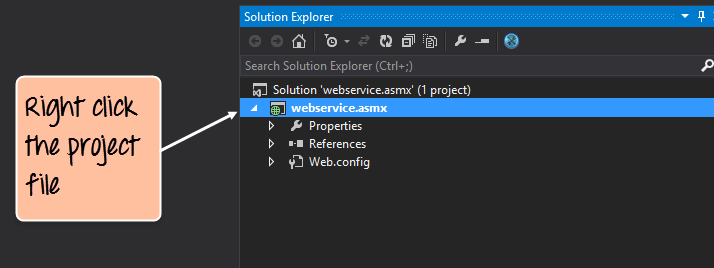
Once done you will see the project file created in your solution explorer in Visual Studio 2013.

[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO7.png)

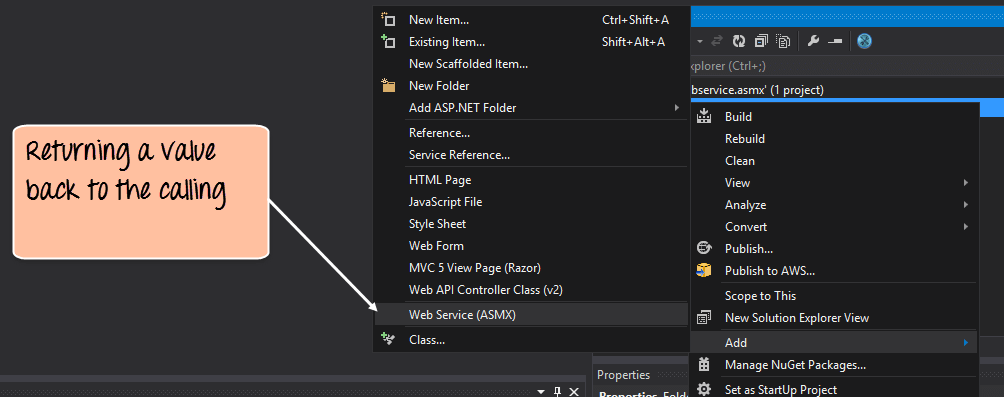
**Step 3)** In this step,

We are going to add a Web service file to our project

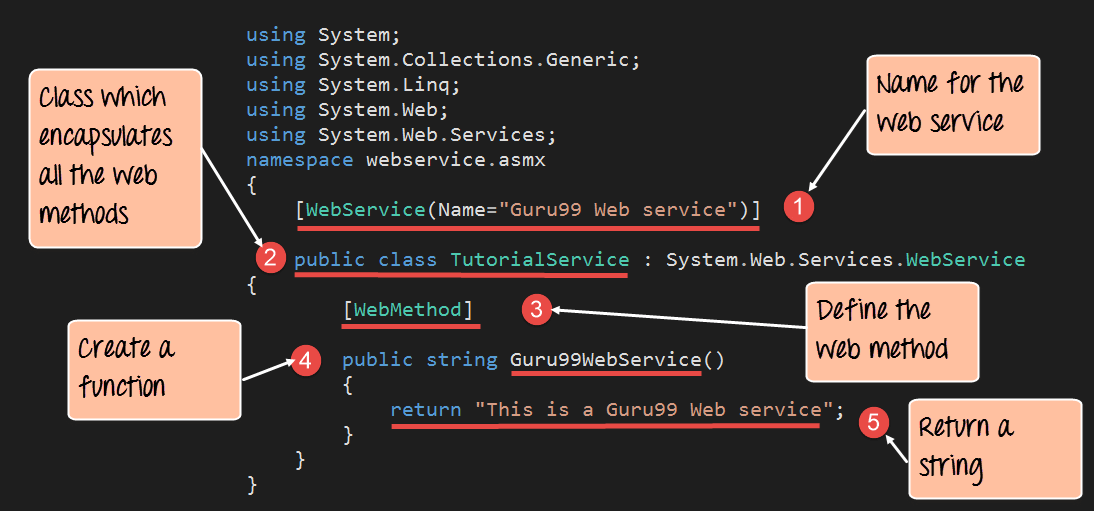
1. First Right-click on the project file as shown below

[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO8.png)

1. Once you right-click on the project file, you have the chance to choose the option "Add->Web Service(ASMX) to add a web service file. Just provide a name of Tutorial Service for the web service name file.

[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO9.png)

**Step 4)** Add the following code to your Tutorial Service asmx file.

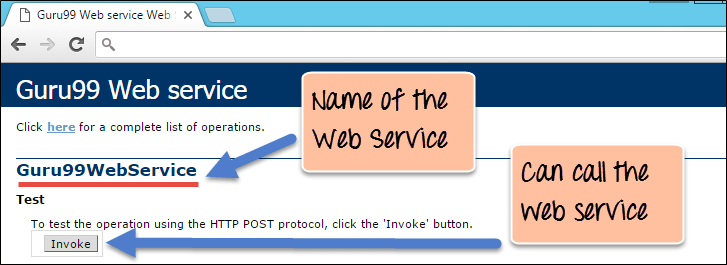
[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO10.png)

**Code Explanation:**

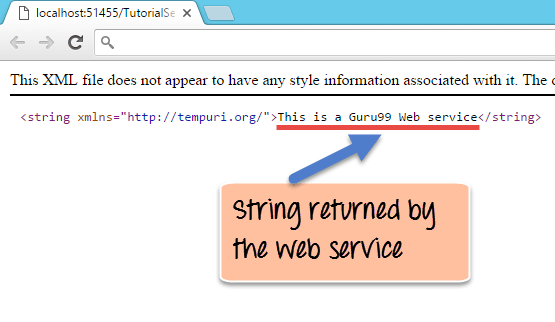
1. This line of code provides a name for your web service file. This is an important step because it gives way for the client application to call the web service via the name of the web service.
2. Normally a class file is used to encapsulate the functionality of a web service. So the class file will have the definition of all the web methods which will provide some functionality to the client application.
3. Here [WebMethod] is known as an attribute which describes a function. The subsequent step creates a function called "Guru99WebService", but with the inclusion of this step of adding a [WebMethod] attribute makes sure that this method can be invoked by a client application. If this attribute is not in place, then the method can never be called by a client application.
4. Here we are defining a function called 'Guru99WebService' which will be used to return a string to the calling client application. This function is a web service which can be called by any client application.
5. We are using the return statement to return the string "This is a Guru99 Web service" to the client application.

If the code is executed successfully, the following Output will be shown when you run your code in the browser.

**Output:**

[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO11.png)

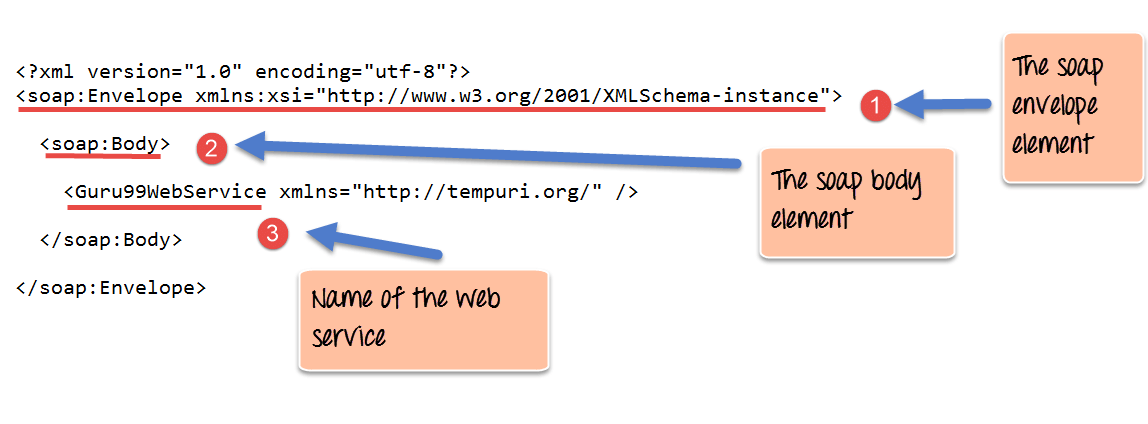
* The output clearly shows that the name of our web service is "Guru99 Web Service" which is the result of giving a name for our web service.
* We can also see that we can to invoke the web service. If we click the Invoke button, we will get the below response in the web browser.

[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO12.png)

The above output,

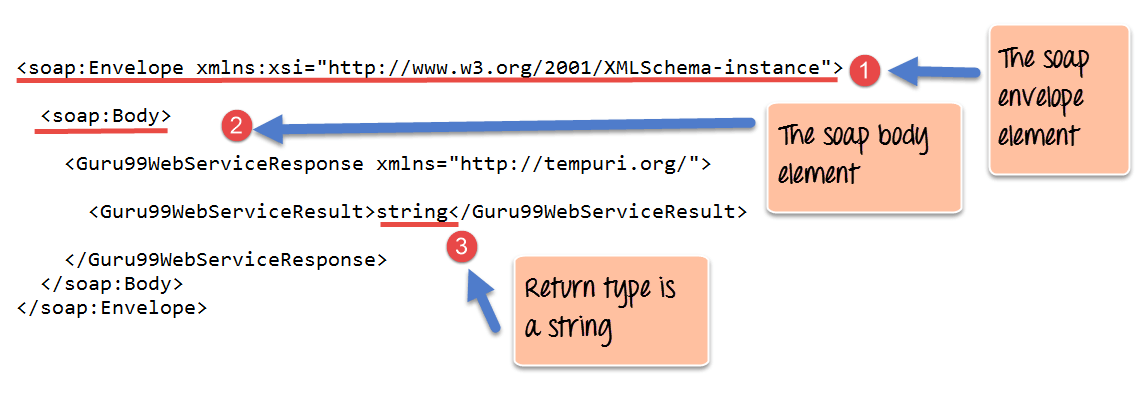
* It clearly shows that by invoking the web method, the string "This is a Gu99 Web service" is returned.
* Visual Studio also allows you to view the SOAP message request and response which is generated when the above web service is called.

The SOAP request which is generated when the web service is called is shown below.

[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO13.png)

**Code Explanation:**

1. The first part of the SOAP message is the envelope element which is what was discussed in the prior chapters. This is the encapsulating element which is present in every SOAP message.
2. The SOAP Body is the next element and contains the actual details of the SOAP message.
3. The third part is the element which specifies that we want to call the service which is called 'Guru99WebService.'

[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO14.png)

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<soap:Body>

<Guru99WebServiceResponse xmlns="http://tempuri.org/">

<Guru99WebServiceResult>string</Guru99WebServiceResult>

</Guru99WebServiceResponse>

</soap:Body>

</soap:Envelope>

**Code Explanation:**

1. The first part of the SOAP message is the envelope element which is what was discussed in the prior chapters. This is the encapsulating element which is present in every SOAP message.
2. The SOAP Body is the next element and contains the actual details of the SOAP message.
3. The interesting part you will see now is the 'string' attribute. This tells the client application that the web service being called returns an object of the type string. This is very useful because if the client application which otherwise would not know what the web service returns.

**Summary**

* SOAP is a protocol which is used to interchange data between applications which are built on different programming languages.
* SOAP is built upon the XML specification and works with the HTTP protocol. This makes it a perfect for usage within web applications.
* The SOAP building blocks consist of a SOAP Message. Each SOAP message consists of an envelope element, a header, and a body element.
* The envelope element is the mandatory element in the SOAP message and is used to encapsulate all of the data in the SOAP message.
* The header element can be used to contain information such as authentication information or the definition of complex data types.
* The body element is the main element which contains the definition of the web methods along with any parameter information if required.

**Deployment of a web application**

* Deploy a web application means to make it ready to be used by its clients This is achieved by:

• structuring the files that constitute the web application in a certain standard way and by

• installing (usually, just moving) it in a certain location of the server

**Cache in web server**

Caching is the term for storing reusable responses in order to make subsequent requests faster. There are many different types of caching available, each of which has its own characteristics. Application caches and memory caches are both popular for their ability to speed up certain responses.

Web caching, the focus of this guide, is a different type of cache. Web caching is a core design feature of the HTTP protocol meant to minimize network traffic while improving the perceived responsiveness of the system as a whole. Caches are found at every level of a content's journey from the original server to the browser.

Web caching works by caching the HTTP responses for requests according to certain rules. Subsequent requests for cached content can then be fulfilled from a cache closer to the user instead of sending the request all the way back to the web server.

## **Benefits**

Effective caching aids both content consumers and content providers. Some of the benefits that caching brings to content delivery are:

* **Decreased network costs**: Content can be cached at various points in the network path between the content consumer and content origin. When the content is cached closer to the consumer, requests will not cause much additional network activity beyond the cache.
* **Improved responsiveness**: Caching enables content to be retrieved faster because an entire network round trip is not necessary. Caches maintained close to the user, like the browser cache, can make this retrieval nearly instantaneous.
* **Increased performance on the same hardware**: For the server where the content originated, more performance can be squeezed from the same hardware by allowing aggressive caching. The content owner can leverage the powerful servers along the delivery path to take the brunt of certain content loads.
* **Availability of content during network interruptions**: With certain policies, caching can be used to serve content to end users even when it may be unavailable for short periods of time from the origin servers.

## **Terminology**

When dealing with caching, there are a few terms that you are likely to come across that might be unfamiliar. Some of the more common ones are below:

* **Origin server**: The origin server is the original location of the content. If you are acting as the web server administrator, this is the machine that you control. It is responsible for serving any content that could not be retrieved from a cache along the request route and for setting the caching policy for all content.
* **Cache hit ratio**: A cache's effectiveness is measured in terms of its cache hit ratio or hit rate. This is a ratio of the requests able to be retrieved from a cache to the total requests made. A high cache hit ratio means that a high percentage of the content was able to be retrieved from the cache. This is usually the desired outcome for most administrators.
* **Freshness**: Freshness is a term used to describe whether an item within a cache is still considered a candidate to serve to a client. Content in a cache will only be used to respond if it is within the freshness time frame specified by the caching policy.
* **Stale content**: Items in the cache expire according to the cache freshness settings in the caching policy. Expired content is "stale". In general, expired content cannot be used to respond to client requests. The origin server must be re-contacted to retrieve the new content or at least verify that the cached content is still accurate.
* **Validation**: Stale items in the cache can be validated in order to refresh their expiration time. Validation involves checking in with the origin server to see if the cached content still represents the most recent version of item.
* **Invalidation**: Invalidation is the process of removing content from the cache before its specified expiration date. This is necessary if the item has been changed on the origin server and having an outdated item in cache would cause significant issues for the client.

There are plenty of other caching terms, but the ones above should help you get started.

## **What Can be Cached?**

Certain content lends itself more readily to caching than others. Some very cache-friendly content for most sites are:

* Logos and brand images
* Non-rotating images in general (navigation icons, for example)
* Style sheets
* General Javascript files
* Downloadable Content
* Media Files

These tend to change infrequently, so they can benefit from being cached for longer periods of time.

Some items that you have to be careful in caching are:

* HTML pages
* Rotating images
* Frequently modified Javascript and CSS
* Content requested with authentication cookies

Some items that should almost never be cached are:

* Assets related to sensitive data (banking info, etc.)
* Content that is user-specific and frequently changed

In addition to the above general rules, it's possible to specify policies that allow you to cache different types of content appropriately. For instance, if authenticated users all see the same view of your site, it may be possible to cache that view anywhere. If authenticated users see a user-sensitive view of the site that will be valid for some time, you may tell the user's browser to cache, but tell any intermediary caches not to store the view.

## **Locations Where Web Content Is Cached**

Content can be cached at many different points throughout the delivery chain:

* **Browser cache**: Web browsers themselves maintain a small cache. Typically, the browser sets a policy that dictates the most important items to cache. This may be user-specific content or content deemed expensive to download and likely to be requested again.
* **Intermediary caching proxies**: Any server in between the client and your infrastructure can cache certain content as desired. These caches may be maintained by ISPs or other independent parties.
* **Reverse Cache**: Your server infrastructure can implement its own cache for backend services. This way, content can be served from the point-of-contact instead of hitting backend servers on each request.

Each of these locations can and often do cache items according to their own caching policies and the policies set at the content origin.

## **Caching Headers**

Caching policy is dependent upon two different factors. The caching entity itself gets to decide whether or not to cache acceptable content. It can decide to cache less than it is allowed to cache, but never more.

The majority of caching behavior is determined by the caching policy, which is set by the content owner. These policies are mainly articulated through the use of specific HTTP headers.

Through various iterations of the HTTP protocol, a few different cache-focused headers have arisen with varying levels of sophistication. The ones you probably still need to pay attention to are below:

* **Expires**: The Expires header is very straight-forward, although fairly limited in scope. Basically, it sets a time in the future when the content will expire. At this point, any requests for the same content will have to go back to the origin server. This header is probably best used only as a fall back.
* **Cache-Control**: This is the more modern replacement for the Expires header. It is well supported and implements a much more flexible design. In almost all cases, this is preferable to Expires, but it may not hurt to set both values. We will discuss the specifics of the options you can set with Cache-Control a bit later.
* **Etag**: The Etag header is used with cache validation. The origin can provide a unique Etag for an item when it initially serves the content. When a cache needs to validate the content it has on-hand upon expiration, it can send back the Etag it has for the content. The origin will either tell the cache that the content is the same, or send the updated content (with the new Etag).
* **Last-Modified**: This header specifies the last time that the item was modified. This may be used as part of the validation strategy to ensure fresh content.
* **Content-Length**: While not specifically involved in caching, the Content-Length header is important to set when defining caching policies. Certain software will refuse to cache content if it does not know in advanced the size of the content it will need to reserve space for.
* **Vary**: A cache typically uses the requested host and the path to the resource as the key with which to store the cache item. The Vary header can be used to tell caches to pay attention to an additional header when deciding whether a request is for the same item. This is most commonly used to tell caches to key by the Accept-Encoding header as well, so that the cache will know to differentiate between compressed and uncompressed content.

A cache server is a dedicated network [server](https://whatis.techtarget.com/definition/server) or service acting as a server that saves Web pages or other Internet content locally. By placing previously requested information in temporary storage, or [cache](https://searchstorage.techtarget.com/definition/cache), a cache server both speeds up access to data and reduces demand on an enterprise's bandwidth. Cache servers also allow users to access content offline, including [rich media](https://whatis.techtarget.com/definition/rich-media) files or other documents. A cache server is sometimes called a "cache engine."

A cache server is almost always also a [proxy server](https://whatis.techtarget.com/definition/proxy-server), which is a server that "represents" users by intercepting their Internet requests and managing them for users. Typically, this is because enterprise resources are being protected by a [firewall](https://searchsecurity.techtarget.com/definition/firewall) server. That server allows outgoing requests to go out but screens all incoming traffic. A proxy server helps match incoming messages with outgoing requests. In doing so, it is in a position to also cache the files that are received for later recall by any user. To the user, the proxy and cache servers are invisible; all Internet requests and returned responses appear to be coming from the addressed place on the Internet. (The proxy is not quite invisible; its [IP](https://searchunifiedcommunications.techtarget.com/definition/Internet-Protocol) address has to be specified as a configuration option to the browser or other protocol program.)