# JEPPIAAR INSTITUTE OF TECHNOLOGY

**"Self Belief | Self Discipline | Self Respect"**

**DEPARTMENT**

**OF**

**COMPUTER SCIENCE AND ENGINEERING**

**LECTURE NOTES**

**CS8651-INTERNET PROGRAMMING**

**(Regulation 2017)**

**Year/Semester: III / 06 CSE**

**2020 – 2021**

**Prepared by**

**Dr. K. Tamilarasi**

**Associate Professor /CSE**

# UNIT V
# INTRODUCTION TO AJAX and WEB SERVICES

**AJAX: Ajax Client Server Architecture-XML Http Request Object-Call Back Methods; Web Services: Introduction- Java web services Basics – Creating, Publishing, Testing and Describing a Web services (WSDL)-Consuming a web service, Database Driven web service from an application –SOAP**.

## 5.1 AJAX: AJAX CLIENT SERVER ARCHITECTURE

AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

- Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
- Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background.
- Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven.

### How AJAX works?

AJAX communicates with the server using XMLHttpRequest object. Let's try to understand the flow of ajax or how ajax works by the image displayed below.

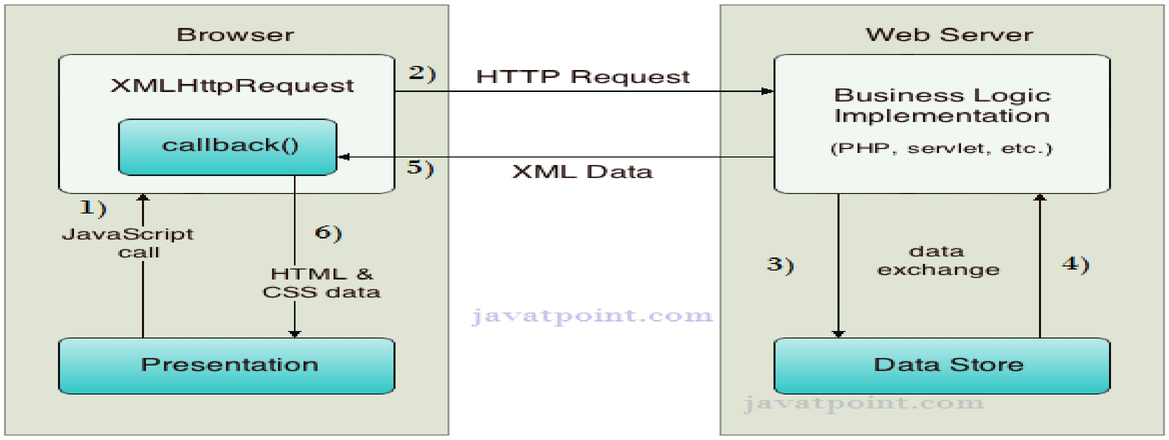


**Fig: 5.1 AJAX Architecture**

XMLHttpRequest object plays a important role.

1. User sends a request from the UI and a javascript call goes to XMLHttpRequest object.
2. HTTP Request is sent to the server by XMLHttpRequest object.
3. Server interacts with the database using JSP, PHP, Servlet, ASP.net etc.
4. Data is retrieved.
5. Server sends XML data or JSON data to the XMLHttpRequest callback function.
6. HTML and CSS data is displayed on the browser.

## 5.2 XML HTTP REQUEST OBJECT

- The XMLHttpRequest object is the key to AJAX. It has been available ever since Internet Explorer 5.5 was released in July 2000, but was not fully discovered until AJAX and Web 2.0 in 2005 became popular.
- XMLHttpRequest (XHR) is an API that can be used by JavaScript, JScript, VBScript, and other web browser scripting languages to transfer and manipulate XML data to and from a webserver using HTTP, establishing an independent connection channel between a webpage's Client-Side and Server-Side.
- The data returned from XMLHttpRequest calls will often be provided by back-end databases. Besides XML, XMLHttpRequest can be used to fetch data in other formats, e.g. JSON or even plain text.
- Listed below are some of the methods and properties that you have to get familiar with.

### 5.2.1 XMLHttpRequest Methods

- **abort()-**Cancels the current request.
- **getAllResponseHeaders()-**Returns the complete set of HTTP headers as a string.
- getResponseHeader( headerName )-Returns the value of the specified HTTP header.

- **open( method, URL )**
- **open( method, URL, async )**
- **open( method, URL, async, userName )**
- **open( method, URL, async, userName, password )**
- Specifies the method, URL, and other optional attributes of a request.
- The method parameter can have a value of "GET", "POST", or "HEAD". Other HTTP methods such as "PUT" and "DELETE" (primarily used in REST applications) may be possible.
- The "async" parameter specifies whether the request should be handled asynchronously or not. "true" means that the script processing carries on after the send() method without waiting for a response, and "false" means that the script waits for a response before continuing script processing.
- **send( content )-**Sends the request.
- **setRequestHeader( label, value )-**Adds a label/value pair to the HTTP header to be sent.

**5.2.3 XMLHttpRequest Properties**

- **onreadystatechange-**An event handler for an event that fires at every state change.
- **readyState-**The readyState property defines the current state of the XMLHttpRequest object.

The following table provides a list of the possible values for the readyState property –

| State | Description |
|---|---|
| 0 | The request is not initialized. |
| 1 | The request has been set up. |
| 2 | The request has been sent. |
| 3 | The request is in process. |
| 4 | The request is completed. |

- **readyState =** 0 After you have created the XMLHttpRequest object, but before you have called the open() method.
- **readyState =** 1 After you have called the open() method, but before you have called send().
- **readyState =** 2 After you have called send().

- **readyState** = 3 After the browser has established a communication with the server, but before the server has completed the response.
- **readyState** = 4 After the request has been completed, and the response data has been completely received from the server.
- **responseText-**Returns the response as a string.
- **responseXML-**Returns the response as XML. This property returns an XML document object, which can be examined and parsed using the W3C DOM node tree methods and properties.
- **Status-**Returns the status as a number (e.g., 404 for "Not Found" and 200 for "OK").
- **Status Text**-Returns the status as a string (e.g., "Not Found" or "OK").

## 5.3 CALL BACK METHODS

The ajaxSuccess( callback ) method attaches a function to be executed whenever an AJAX request completes successfully. This is an Ajax Event.

### Syntax

Here is the simple syntax to use this method −

```
$(document).ajaxSuccess( callback )
```

### Parameters

Here is the description of all the parameters used by this method −

- **callback** − The function to execute. The event object, XMLHttpRequest, and settings used for that request are passed as arguments to the callback.

### Example

Assuming we have following HTML content in result.html file −

**<h1>THIS IS RESULT...</h1>**

Following is a simple example a simple showing the usage of this method.

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
    </script>
        <script type = "text/javascript" language = "javascript">
```

```
$(document).ready(function() {
   /* Global variable */
   var count = 0;
   $("#driver").click(function(event){
      $('#stage0').load('result.html');
   });
   /* Gets called when request starts */
   $(document).ajaxStart(function(){
      count++;
      $("#stage1").html("<h1>Starts, Count :" + count + "</h1>");
   });
   /* Gets called when request is sent */
   $(document).ajaxSend(function(evt, req, set){
      count++;
      $("#stage2").html("<h1>Sends, Count :" + count + "</h1>");
      $("#stage2").append("<h1>URL :" + set.url  + "</h1>");
   });
   /* Gets called when request completes */
   $(document).ajaxComplete(function(event,request,settings){
      count++;
      $("#stage3").html("<h1>Completes,Count:" + count + "</h1>");
   });
   /* Gets called when request is stopped */
   $(document).ajaxStop(function(event,request,settings){
      count++;
      $("#stage4").html("<h1>Stops, Count :" + count + "</h1>");
```

```
        });
        /* Gets called when all request completes successfully */
        $(document).ajaxSuccess(function(event,request,settings){
          count++;
          $("#stage5").html("<h1>Success,Count :" + count + "</h1>");
        });
     });
   </script>
 </head>
 <body>
   <p>Click on the button to load result.html file:</p>
   <div id = "stage0" style = "background-color:blue;">
     STAGE - 0
   </div>
   <div id = "stage1" style = "background-color:blue;">
     STAGE - 1
   </div>
   <div id = "stage2" style = "background-color:blue;">
     STAGE - 2
   </div>
   <div id = "stage3" style = "background-color:blue;">
     STAGE - 3
   </div>
   <div id = "stage4" style = "background-color:blue;">
     STAGE - 4
   </div>
```

```
    <div id = "stage5" style = "background-color:blue;">
       STAGE - 5
    </div>
    <input type = "button" id = "driver" value="Load Data" />
  </body>
</html>
```

**Output:**

Click on the button to load result.html file −

STAGE - 0

STAGE - 1

STAGE – 2

**5.4 WEB SERVICES-INTRODUCTION**

- A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language—Java can talk with Perl; Windows applications can talk with Unix applications.
- Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.
- Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.
- A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

To summarize, a complete web service is, therefore, any service that −

- Is available over the Internet or private (intranet) networks
- Uses a standardized XML messaging system
- Is not tied to any one operating system or programming language
- Is self-describing via a common XML grammar
- Is discoverable via a simple find mechanism

**Components of Web Services**

The basic web services platform is XML + HTTP. All the standard web services work using the following components −

- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language)

**How Does a Web Service Work?**

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of −

- XML to tag the data
- SOAP to transfer a message
- WSDL to describe the availability of service.

**Example**

Consider a simple account-management and order processing system. The accounting personnel use a client application built with Visual Basic or JSP to create new accounts and enter new customer orders.

The processing logic for this system is written in Java and resides on a Solaris machine, which also interacts with a database to store information.

The steps to perform this operation are as follows −

- The client program bundles the account registration information into a SOAP message.
- This SOAP message is sent to the web service as the body of an HTTP POST request.
- The web service unpacks the SOAP request and converts it into a command that the application can understand.

- The application processes the information as required and responds with a new unique account number for that customer.
- Next, the web service packages the response into another SOAP message, which it sends back to the client program in response to its HTTP request.
- The client program unpacks the SOAP message to obtain the results of the account registration process.

## 5.5 WSDL

WSDL is an XML-based file which basically tells the client application what the web service does. It is known as the Web Services Description Language(WSDL).

The WSDL file is used to describe in a nutshell what the web service does and gives the client all the information required to connect to the web service and use all the functionality provided by the web service.

### 5.5.1 WSDL Elements

The WSDL file contains the following main parts

1. The **<types>** tag is used to define all the complex datatypes, which will be used in the message exchanged between the client application and the web service. This is an important aspect of the client application, because if the web service works with a complex data type, then the client application should know how to process the complex data type. Data types such as float, numbers, and strings are all simple data types, but there could be structured data types which may be provided by the web service.

For example, there could be a data type called EmployeeDataType which could have 2 elements called "EmployeeName" of type string and "EmployeeID" of type number or integer. Together they form a data structure which then becomes a complex data type.

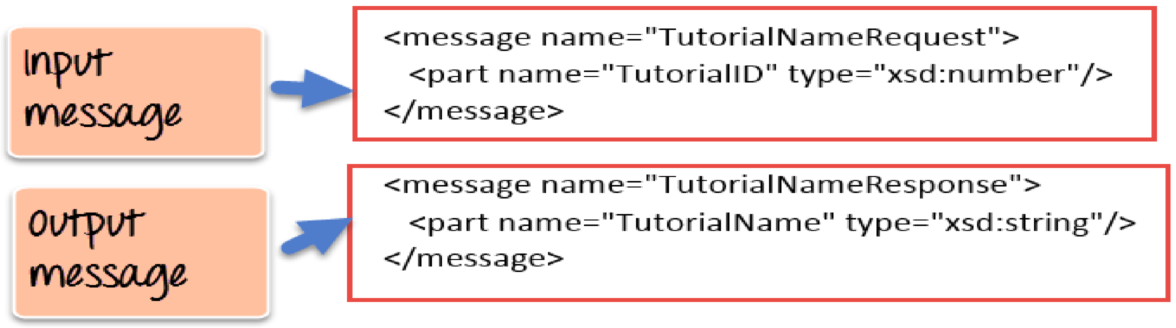
2. The **<messages>** tag is used to define the message which is exchanged between the client application and the web server. These messages will explain the input and output operations which can be performed by the web service. An example of a message can be a message which accepts the EmployeeID of an employee, and the output message can be the name of the employee based on the EmpoyeeID provided.
3. The **<portType>** tag is used to encapsulate every input and output message into one logical operation. So there could be an operation called "GetEmployee" which combines the input message of accepting the EmployeeID from a client application and then sending the EmployeeName as the output message.
4. The **<binding>** tag is used to bind the operation to the particular port type. This is so that when the client application calls the relevant port type, it will then be able to access the operations which are bound to this port type. Port types are just like interfaces. So if

a client application needs to use a web service they need to use the binding information to ensure that they can connect to the interface provided by that web service.

5. The **<service>** tag is a name given to the web service itself. Initially, when a client application makes a call to the web service, it will do by calling the name of the web service. For example, a web service can be located at an address such as **http://localhost/Guru99/Tutorial.asmx** . The service tag will actually have the URL defined as **http://localhost/Guru99/Tutorial.asmx**, which will actually tell the client application that there is a web service available at this location.

### 5.5.2 WSDL Message Part

- The WSDL consists of a section called "messages" which is denoted by the **<message>** element.
- This element is basically used to describe the data that gets exchanged between the web service and the client application.
- Each web service will always have 2 types of messages,
  - One is for the input of the web service, and the other is for the output of the web service.
  - The input is used to describe the parameters which are accepted by the web service. This is an important aspect of the client application so that it knows the values to be sent as parameters to the web service.
  - The other type of message is the output message which tells what results are provided by the web service.
- Each message, in turn, will have a **<part>** element which is used to describe the parameter used by the input and output message.
- Below is a simple example, of what a message for a web service looks like. The functionality of the web service is to provide the name of a "Tutorial" once a "Tutorial ID" is submitted as a parameter to the web service.



```
<message name="TutorialNameRequest">
  <part name="TutorialID" type="xsd:number"/>
</message>
```

```
<message name="TutorialNameResponse">
  <part name="TutorialName" type="xsd:string"/>
</message>
```

**Fig: 5.2 WSDL Message Part**

1. As we can see the web service has 2 messages, one for the input and the other for the output.

2. The input message is known as TutorialNameRequest which has one parameter called TutorialID. This parameter is of the type number which is specified by the xsd:number type
3. The output message is known as TutorialNameResponse which has one parameter called TutorialName. This parameter is of the type string which is specified by the xsd:string type

### 5.5.3 Creating WSDL File

- The WSDL file gets created whenever a web service is built in any programming language.
- Since the WSDL file is pretty complicated to be generated from plain scratch, all editors such as Visual Studio for .Net and Eclipse for Java automatically create the WSDL file.
- Below is an example of a WSDL file created in Visual Studio.

```
<?xml version="1.0"?>
<definitions name="Tutorial"
               targetNamespace=http://Guru99.com/Tutorial.wsdl
        xmlns:tns=http://Guru99.com/Tutorial.wsdl
        xmlns:xsd1=http://Guru99.com/Tutorial.xsd
        xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
        xmlns="http://schemas.xmlsoap.org/wsdl/">
   <types>
               <schema targetNamespace=http://Guru99.com/Tutorial.xsd
        xmlns="http://www.w3.org/2000/10/XMLSchema">

        <element name="TutorialNameRequest">
               <complexType>
               <all>
                       <element name="TutorialName" type="string"/>
                </all>
            </complexType>
       </element>
       <element name="TutorialIDRequest">
               <complexType>
               <all>
                       <element name="TutorialID" type="number"/>
                </all>
            </complexType>
       </element>
    </schema>
 </types>
 <message name="GetTutorialNameInput">
```

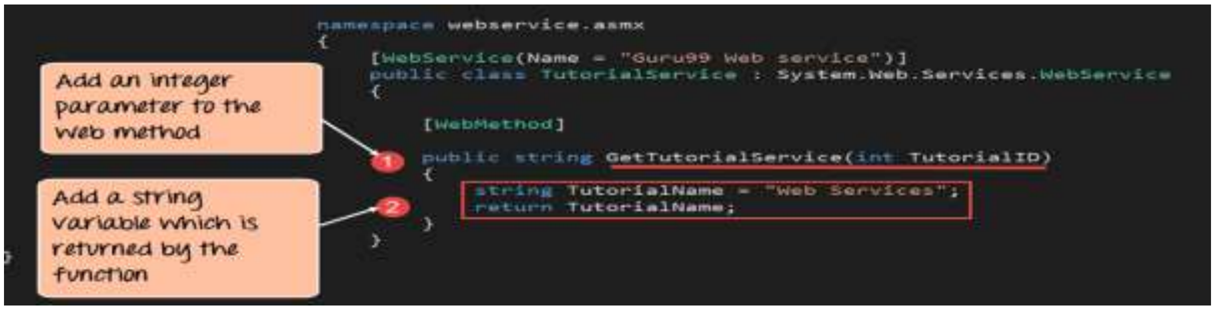
```
       <part name="body" element="xsd1:TutorialIDRequest"/>
 </message>
 <message name="GetTutorialNameOutput">
       <part name="body" element="xsd1:TutorialNameRequest"/>
 </message>
 <portType name="TutorialPortType">
       <operation name="GetTutorialName">
       <input message="tns:GetTutorialNameInput"/>
        <output message="tns:GetTutorialNameOutput"/>
    </operation>
  </portType>
  <binding name="TutorialSoapBinding" type="tns:TutorialPortType">
  <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetTutorialName">
              <soap:operation
soapAction="http://Guru99.com/GetTutorialName"/>
              <input>
              <soap:body use="literal"/>
           </input>
        <output>
   <soap:body use="literal"/>
 </output>
 </operation>
 </binding>

 <service name="TutorialService">
       <documentation>TutorialService</documentation>
    <port name="TutorialPort" binding="tns:TutorialSoapBinding">
       <soap:address location="http://Guru99.com/Tutorial"/>
    </port>
 </service>
</definitions>
```

### 5.5.4 Publishing the Web Service Example

- An example of how we can publish a web service and consume it by using Visual Studio.
- **Step 1)** The first step is to create your web service. The detailed steps of how the Asp.Net web project and a web service is created. The key part is to enter the below code in the Web services file.

```
namespace webservic asmx
```

```
{
        [WebService(Name = "Guru99 Web service")]
        public class TutorialService : System.Web.Services.WebService
        {
                [WebMethod]
                public string GetTutorialService(int TutorialID)
                {
                        string TutorialName = "Web Services";
                        return TutorialName;
                }
        }
}
```


Add an integer parameter to the web method
Add a string variable which is returned by the function
1
2

```
namespace webservice.asmx
{
    [WebService(Name = "Guru99 Web service")]
    public class TutorialService : System.Web.Services.WebService
    {
        [WebMethod]
        public string GetTutorialService(int TutorialID)
        {
            string TutorialName = "Web Services";
            return TutorialName;
        }
    }
```

**Fig: 5.3 creating web Service**

**Code Explanation:**

1. Here we are creating a WebMethod called "Guru99WebService." In this web method, we are including an integer parameter which needs to be passed whenever this web method is called.

2. Next we are defining a variable called "TutorialName" which will hold the string value of "Web Services." This is the value which will be returned when the web service is called.

**Step 2)** Once we have defined the web services file, the next step is to create a client project which will consume this web service.

- Let's create a simple console application which will call this web service, invoke the "Guru99WebService" and then display the output of the web method in the console log screen. Follow the below steps to create a console application.
- Right-click the Visual Studio solution file and choose the option Add->New project

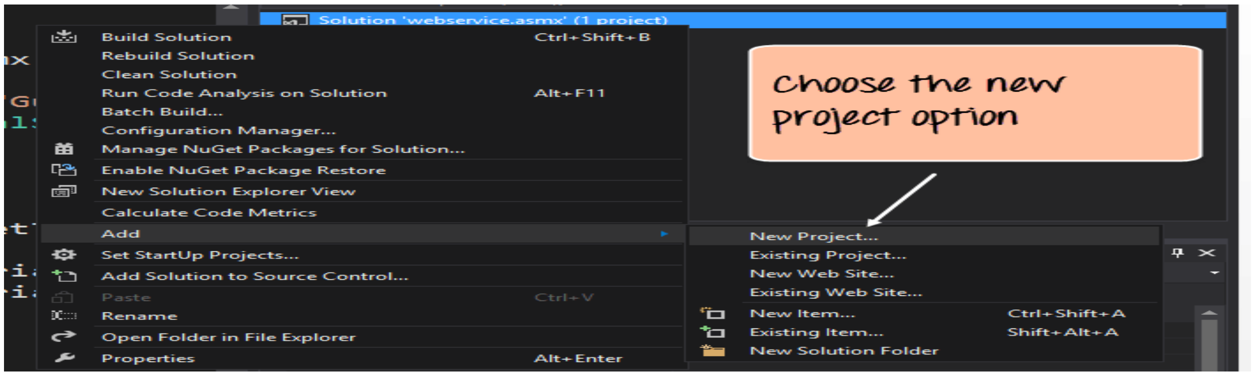
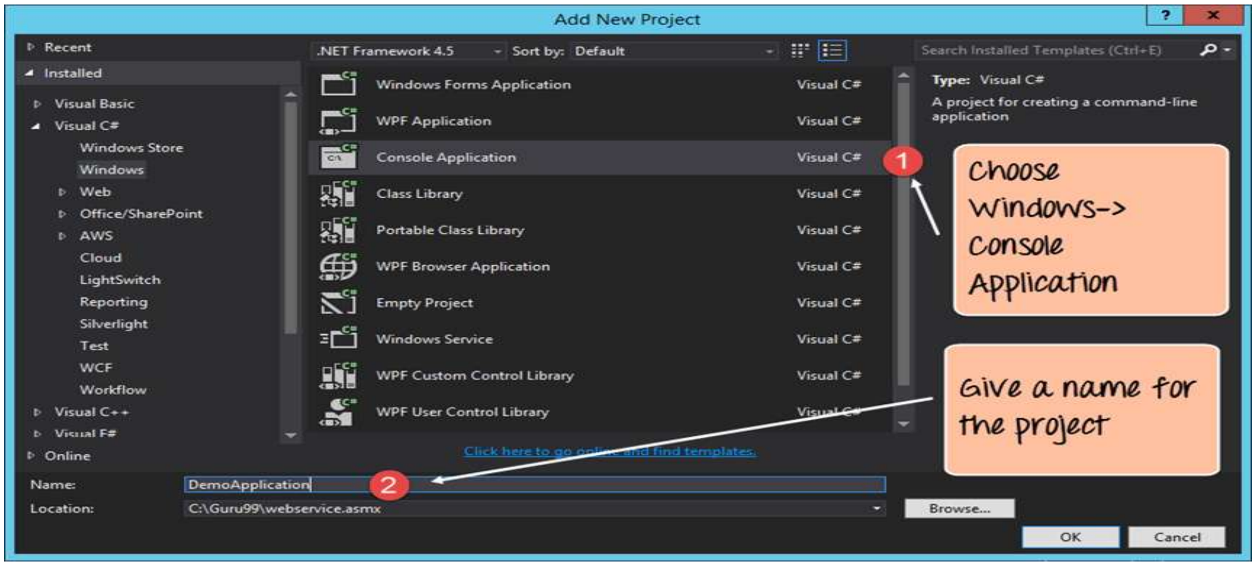

Solution 'webservice.asmx' (1 project)
Build Solution Ctrl+Shift+B
Rebuild Solution
Clean Solution
Run Code Analysis on Solution Alt+F11
Batch Build...
Configuration Manager...
Manage NuGet Packages for Solution...
Enable NuGet Package Restore
New Solution Explorer View
Calculate Code Metrics
Add
Set StartUp Projects...
Add Solution to Source Control...
Paste Ctrl+V
Rename
Open Folder in File Explorer
Properties Alt+Enter
Choose the new project option
New Project...
Existing Project...
New Web Site...
Existing Web Site...
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Solution Folder

**Fig 5.4 Creating a console Application**

**Step3)** In this step,

1. Ensure to first choose the Visual C# Windows option. Then choose the option of creating a console application.
2. Give a name for your project which in our case has been given as "DemoApplication."


Add New Project
Recent
Installed
Visual Basic
Visual C#
Windows Store
Windows
Web
Office/SharePoint
AWS
Cloud
LightSwitch
Reporting
Silverlight
Test
WCF
Workflow
Visual C++
Visual F#
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (Ctrl+E)
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Class Library Visual C#
Portable Class Library Visual C#
WPF Browser Application Visual C#
Empty Project Visual C#
Windows Service Visual C#
WPF Custom Control Library Visual C#
WPF User Control Library Visual C#
Click here to go online and find templates.
1
Choose Windows-> Console Application
Give a name for the project
2
Browse...
OK
Cancel

Type: Visual C#
A project for creating a command-line application

Name: DemoApplication
Location: C:\Guru99\webservice.asmx

**Fig: 5.5 Naming the Project**

- After you click the OK button in the above screen, you will be able to see the project in the Solution explorer in Visual Studio.

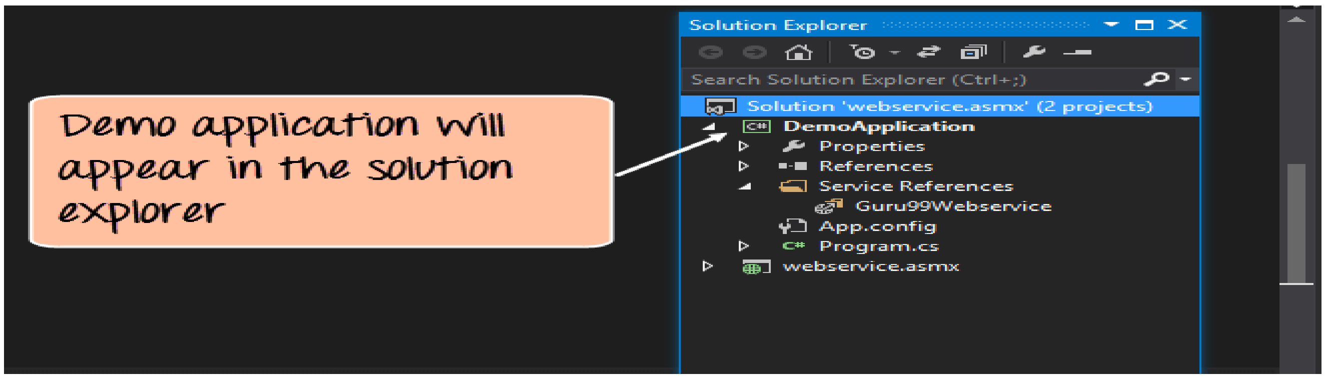

Demo application will appear in the solution explorer
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'webservice.asmx' (2 projects)
DemoApplication
Properties
References
Service References
Guru99Webservice
App.config
Program.cs
webservice.asmx

**Fig: 5.6 Displaying the Application**

**Step 4)** In this step, you be setting the DemoApplication Console application as the startup project. This is done to ensure that this application launches first when the entire Visual Studio project is run. This Console application will, in turn, call the web service which will be automatically launched by Visual Studio.

- To complete this step, right-click the DemoApplication project and choose the option "Set as StartUp Project."

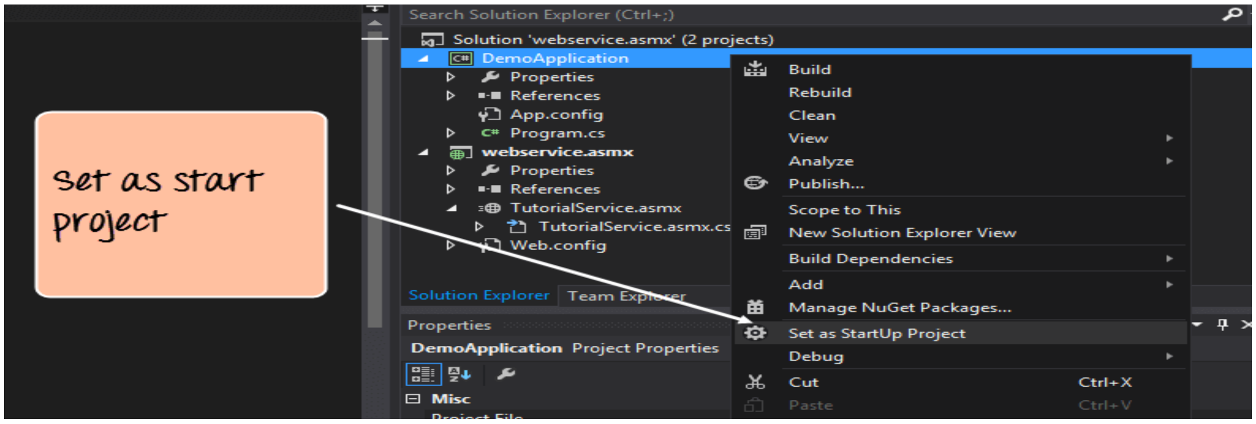


**Fig: 5.7 Start the New Project**

**Step 5)** The next step is to add the service reference of our "Guru99Webservice" to our console application. This is done so that the DemoApplication can reference the web service and all of the web methods in the web service.

- To do this, right-click the DemoApplication project file and choose the menu option Add->Service Reference.

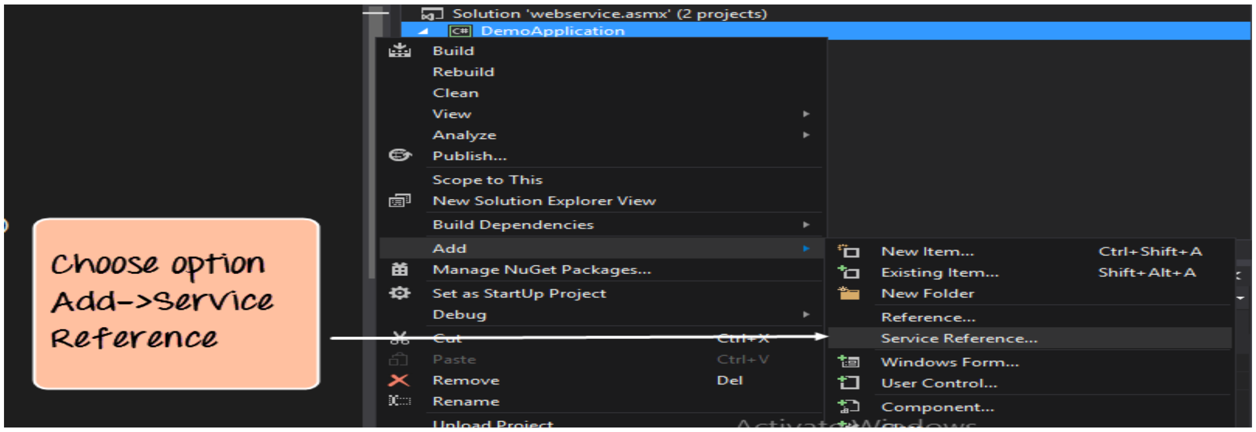


**Fig: 5.8 Add the Service Reference**

**Step 6)** In this step, we will provide the different values which are required to add our service reference

1. Firstly we need to choose our discover option. This option will automatically pick up the WSDL file for our TutorialService web service.
2. Next, we should give a name for our service reference. In our case, we are giving it a name of Guru99Webservice.
3. Then we need to expand on the TutorialService.asmx option so that we can have the ability to see the 'GetTutorialService' method on the right-hand side. Here TutorialService.asmx is the name of our Visual Studio .Net file which contains the code for our web service.
4. We will then see our Web method which we had in our web service known as "GetTutorialService"

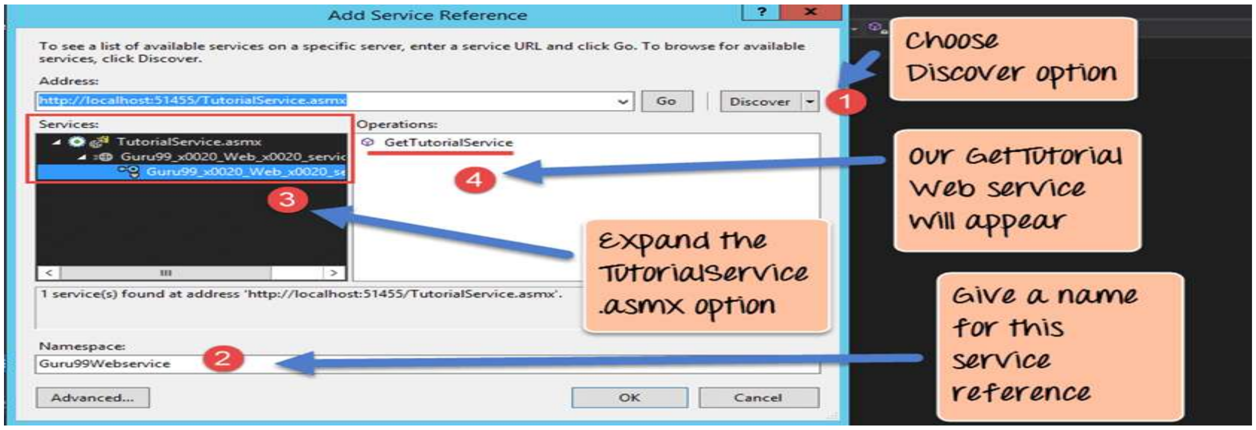

Add Service Reference
To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.
Address:
http://localhost:51455/TutorialService.asmx
Go
Discover
Services:
TutorialService.asmx
Guru99_x0020_Web_x0020_servic
Guru99_x0020_Web_x0020_se
Operations:
GetTutorialService
1 service(s) found at address 'http://localhost:51455/TutorialService.asmx'.
Namespace:
Guru99Webservice
Advanced...
OK
Cancel
1
2
3
4
Choose Discover option
Our GetTutorial Web service will appear
Expand the TutorialService .asmx option
Give a name for this service reference

**Fig: 5.9 Displaying the Web Service Reference**

- When we click on the 'OK' button, all of the required code to access this web service will be added to our DemoApplication Console application as shown below.
- The screenshot shows that the "Guru99Webservice" was successfully added to our console application.

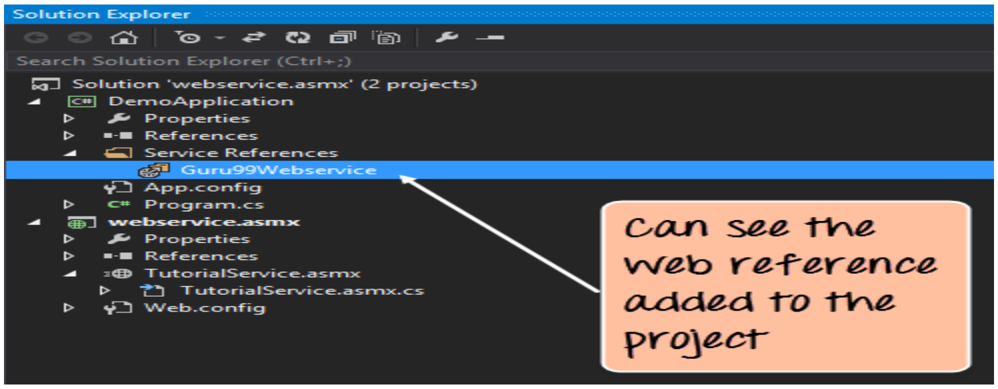
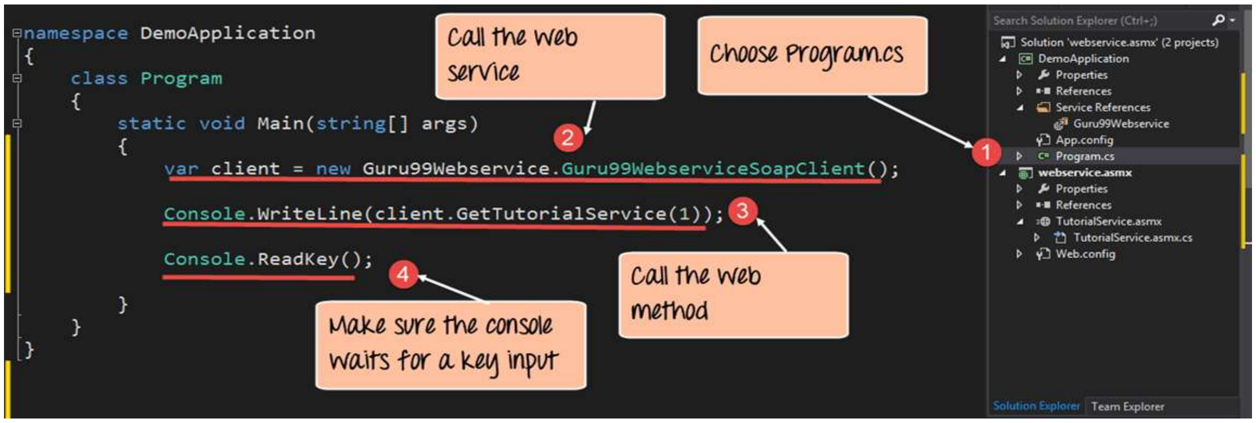


**Fig: 5.10 Web reference added to Project**

**Step 7)** The next step is to add the code to our console application to access the web method in our web service. Open the Program.cs code file which comes automatically with the console application and add the below code



```
namespace DemoApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            var client = new Guru99Webservice.Guru99WebserviceSoapClient();

            Console.WriteLine(client.GetTutorialService(1));

            Console.ReadKey();
        }
    }
}
```

**Fig: 5.11 Call the web service**

```
namespace DemoApplication
{
        class Program
        {
                static void Main(string[ ] args)
                {
                        var client = new
Guru99Webservice.Guru99WebserviceSoapClient();
```

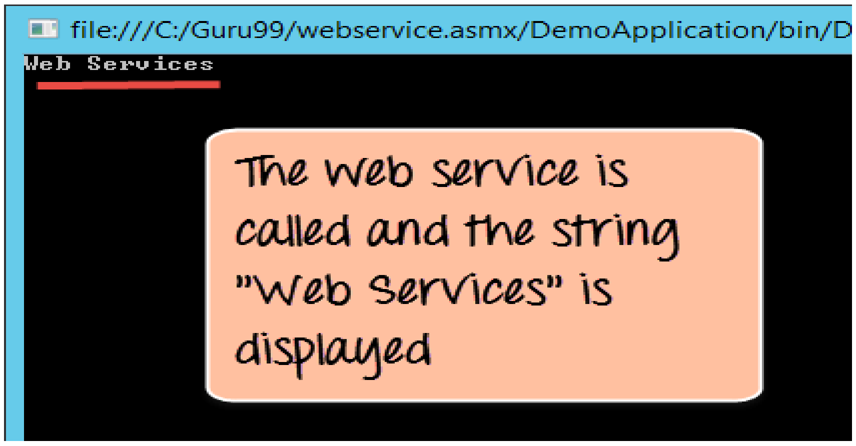
```
                        Console.WriteLine(client.GetTutorialService(1));

                        Console.ReadKey();
                }
        }
}
```

**Code Explanation:-**

1. The first part is to choose the Program.cs file. This is the main file which is created by Visual Studio when a console application is created. This file is what gets executed when the console application (in our case demo application) is executed.
2. We then create a variable called "client" which will be set to an instance of our Service reference which was created in an earlier step. In our case, the service reference is 'Guru99Webservice.Guru99WebserviveSoapClient()'
3. We are then calling our Webmethod 'GetTutorialService' in the TutorialService web service Remember that our GetTutorialService' method accepts an integer parameter, so we are just passing an integer parameter to the web method.
4. This final line is just to ensure the console log screen remains active so that we can view the output. This command will just wait for some input from the user.

**Output**

- When all of the above steps are followed, and the DemoApplication is run the below output will be displayed.


file:///C:/Guru99/webservice.asmx/DemoApplication/bin/D
The web service is called and the string "Web Services" is displayed

```
Web Services
```

**Fig: 5.11 WSDL Output**

- From the output, we can clearly see that the DemoApplication calls our Web service and that the string returned by the Web service is displayed in our Console log.

## 5.6 SOAP

SOAP is an acronym for Simple Object Access Protocol. It is an XML-based messaging protocol for exchanging information among computers. SOAP is an application of the XML specification.

- SOAP is a communication protocol designed to communicate via Internet.
- SOAP can extend HTTP for XML messaging.
- SOAP provides data transport for Web services.
- SOAP can exchange complete documents or call a remote procedure.
- SOAP can be used for broadcasting a message.
- SOAP is platform- and language-independent.
- SOAP is the XML way of defining what information is sent and how.
- SOAP enables client applications to easily connect to remote services and invoke remote methods.
- Although SOAP can be used in a variety of messaging systems and can be delivered via a variety of transport protocols, the initial focus of SOAP is remote procedure calls transported via HTTP.
- Other frameworks including CORBA, DCOM, and Java RMI provide similar functionality to SOAP, but SOAP messages are written entirely in XML and are therefore uniquely platform- and language-independent.

**5.6.1 SOAP Message Structure**

A SOAP message is an ordinary XML document containing the following elements −

- Envelope − Defines the start and the end of the message. It is a mandatory element.
- Header − Contains any optional attributes of the message used in processing the message, either at an intermediary point or at the ultimate end-point. It is an optional element.
- Body − Contains the XML data comprising the message being sent. It is a mandatory element.
- Fault − An optional Fault element that provides information about errors that occur while processing the message.
- The following block depicts the general structure of a SOAP message –

```
<?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-ENV:Header>
      ...
      ...
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      ...
      ...
      <SOAP-ENV:Fault>
         ...
         ...
      </SOAP-ENV:Fault>
      ...
   </SOAP-ENV:Body>
</SOAP_ENV:Envelope>
```

**5.6.2 SOAP-Envelope**

The SOAP envelope indicates the start and the end of the message so that the receiver knows when an entire message has been received. The SOAP envelope solves the problem of knowing when you are done receiving a message and are ready to process it. The SOAP envelope is therefore basically a packaging mechanism.

- Every SOAP message has a root Envelope element.
- Envelope is a mandatory part of SOAP message.
- Every Envelope element must contain exactly one Body element.
- If an Envelope 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.
- The envelope changes when SOAP versions change.
- The SOAP envelope is specified using the *ENV* namespace prefix and the Envelope element.
- The optional SOAP encoding is also specified using a namespace name and the optional *encodingStyle* element, which could also point to an encoding style other than the SOAP one.
- 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 *VersionMismatch* fault if it receives a message that does not include the v1.2 envelope namespace.

**V1.2-Compliant SOAP Message**

```
<?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">
  ...
  Message information goes here
  ...
</SOAP-ENV:Envelope>
```

**SOAP with HTTP POST**

- The following example illustrates the use of a SOAP message within an HTTP POST operation, which sends the message to the server.
- It shows the namespaces for the envelope schema definition and for the schema definition of the encoding rules. T
- he OrderEntry reference in the HTTP header is the name of the program to be invoked at the tutorialspoint.com website.

```
POST /OrderEntry HTTP/1.1
Host: www.tutorialspoint.com
Content-Type: application/soap;  charset="utf-8"
Content-Length: nnnn

<?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">
  ...
  Message information goes here
  ...
</SOAP-ENV:Envelope>
```

**5.6.3 SOAP-Header**

- The optional Header element offers a flexible framework for specifying additional application-level requirements. For example, the Header element can be used to specify a digital signature for password-protected services. Likewise, it can be used to specify an account number for pay-per-use SOAP services.
- It is an optional part of a SOAP message.
- Header elements can occur multiple times.
- Headers are intended to add new features and functionality.
- The SOAP header contains header entries defined in a namespace.

- The header is encoded as the first immediate child element of the SOAP envelope.
- When multiple headers are defined, all immediate child elements of the SOAP header are interpreted as SOAP header blocks.

SOAP Header Attributes

A SOAP Header can have the following two attributes −

**Actor attribute**

- The SOAP protocol defines a message path as a list of SOAP service nodes.
- Each of these intermediate nodes can perform some processing and then forward the message to the next node in the chain.
- By setting the Actor attribute, the client can specify the recipient of the SOAP header.

**MustUnderstand attribute**

- It indicates whether a Header element is optional or mandatory. If set to true, the recipient must understand and process the Header attribute according to its defined semantics, or return a fault.
- The following example shows how to use a Header in a SOAP message.

```
<?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-ENV:Header>
      <t:Transaction
         xmlns:t = "http://www.tutorialspoint.com/transaction/"
         SOAP-ENV:mustUnderstand = "true">5
      </t:Transaction>
   </SOAP-ENV:Header>
   ...
   ...
</SOAP-ENV:Envelope>
```

**5.6.4 SOAP-Body**

- The SOAP body is a mandatory element that contains the application-defined XML data being exchanged in the SOAP message. The body must be contained within the envelope and must follow any headers that might be defined for the message.
- The body is defined as a child element of the envelope, and the semantics for the body are defined in the associated SOAP schema.

- The body contains mandatory information intended for the ultimate receiver of the message. For example −

```
<?xml version = "1.0"?>
<SOAP-ENV:Envelope>
   ........
   <SOAP-ENV:Body>
      <m:GetQuotation xmlns:m = "http://www.tp.com/Quotation">
         <m:Item>Computers</m:Item>
      </m:GetQuotation>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- The example above requests a quotation of computer sets. Note that the m:GetQuotation and the Item elements above are application-specific elements. They are not a part of the SOAP standard.
- Here is the response to the above query −

```
<?xml version = "1.0"?>
<SOAP-ENV:Envelope>
   ........
   <SOAP-ENV:Body>
      <m:GetQuotationResponse               xmlns:m               =
"http://www.tp.com/Quotation">
         <m:Quotation>This is Qutation</m:Quotation>
      </m:GetQuotationResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Normally, the application also defines a schema to contain semantics associated with the request and response elements.
- The *Quotation* service might be implemented using an EJB running in an application server; if so, the SOAP processor would be responsible for mapping the body information as parameters into and out of the EJB implementation of the *GetQuotationResponse* service. The SOAP processor could also be mapping the body information to a .NET object, a CORBA object, a COBOL program, and so on.

### 5.6.5 SOAP-Fault

- If an error occurs during processing, the response to a SOAP message is a SOAP fault element in the body of the message, and the fault is returned to the sender of the SOAP message.

- The SOAP fault mechanism returns specific information about the error, including a predefined code, a description, and the address of the SOAP processor that generated the fault.
- A SOAP message can carry only one fault block.
- Fault is an optional part of a SOAP message.
- For HTTP binding, a successful response is linked to the 200 to 299 range of status codes.
- SOAP Fault is linked to the 500 to 599 range of status codes.

**Sub-elements of Fault**

The SOAP Fault has the following sub elements –

| Sr.No | Sub-element & Description |
|---|---|
| 1 | <faultCode><br>It is a text code used to indicate a class of errors. See the next Table for a listing of predefined fault codes. |
| 2 | <faultString><br>It is a text message explaining the error. |
| 3 | <faultActor><br>It is a text string indicating who caused the fault. It is useful if the SOAP message travels through several nodes in the SOAP message path, and the client needs to know which node caused the error. A node that does not act as the ultimate destination must include a faultActor element. |
| 4 | <detail><br>It is an element used to carry application-specific error messages. The detail element can contain child elements called detail entries. |

**SOAP Fault Codes**

The faultCode values defined below must be used in the *faultcode* element while describing faults.

| Sr.No | Error & Description |
|---|---|
| 1 | SOAP-ENV:VersionMismatch<br>Found an invalid namespace for the SOAP Envelope element. |
| 2 | SOAP-ENV:MustUnderstand<br>An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood. |
| 3 | SOAP-ENV:Client<br>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 |

**SOAP Fault Example**

The following code is a sample Fault. The client has requested a method named *ValidateCreditCard*, but the service does not support such a method. This represents a client request error, and the server returns the following SOAP response −

```
<?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 (ValidateCreditCard) in class (examplesCreditCard) at
          /usr/local/ActivePerl-5.6/lib/site_perl/5.6.0/SOAP/Lite.pm line 1555.
      </faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**5.6.7 SOAP-Encoding**

SOAP includes a built-in set of rules for encoding data types. It enables the SOAP message to indicate specific data types, such as integers, floats, doubles, or arrays.

- SOAP data types are divided into two broad categories − scalar types and compound types.
- Scalar types contain exactly one value such as a last name, price, or product description.
- Compound types contain multiple values such as a purchase order or a list of stock quotes.
- Compound types are further subdivided into arrays and structs.
- The encoding style for a SOAP message is set via the *SOAP-ENV:encodingStyle* attribute.
- To use SOAP 1.1 encoding, use the value http://schemas.xmlsoap.org/soap/encoding/

- To use SOAP 1.2 encoding, use the value http://www.w3.org/2001/12/soap-encoding
- Latest SOAP specification adopts all the built-in types defined by XML Schema. Still, SOAP maintains its own convention for defining constructs not standardized by XML Schema, such as arrays and references.

## Scalar Types

- For scalar types, SOAP adopts all the built-in simple types specified by the XML Schema specification. This includes strings, floats, doubles, and integers.
- The following table lists the main simple types, excerpted from the XML Schema Part 0 − Primer http://www.w3.org/TR/2000/WD-xmlschema-0-20000407/
- For example, here is a SOAP response with a double data type –

| **Simple Types Built-In to XML Schema** | |
|---|---|
| **Simple Type** | **Example** |
| string | Confirm this is electric. |
| boolean | true, false, 1, 0. |
| float | -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN. |
| double | -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN. |
| decimal | -1.23, 0, 123.4, 1000.00. |
| binary | 100010 |
| integer | -126789, -1, 0, 1, 126789. |

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd = "http://www.w3.org/2001/XMLSchema">

   <SOAP-ENV:Body>
      <ns1:getPriceResponse
```

```
         xmlns:ns1 = "urn:examples:priceservice"
         SOAP-ENV:encodingStyle    =    "http://www.w3.org/2001/12/soap-
encoding">
         <return xsi:type = "xsd:double">54.99</return>
      </ns1:getPriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Compound Types**

- SOAP arrays have a very specific set of rules, which require that you specify both the element type and array size. SOAP also supports multidimensional arrays, but not all SOAP implementations support multidimensional functionality.
- To create an array, you must specify it as an *xsi:type* of array. The array must also include an *arrayType* attribute. This attribute is required to specify the data type for the contained elements and the dimension(s) of the array.
- For example, the following attribute specifies an array of 10 double values −

arrayType = "xsd:double[10]"

- In contrast, the following attribute specifies a two-dimensional array of strings −

arrayType = "xsd:string[5,5]"

- Here is a sample SOAP response with an array of double values −

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd = "http://www.w3.org/2001/XMLSchema">

   <SOAP-ENV:Body>
      <ns1:getPriceListResponse
         xmlns:ns1 = "urn:examples:pricelistservice"
         SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-
encoding">

         <return xmlns:ns2 = "http://www.w3.org/2001/09/soap-encoding"
            xsi:type = "ns2:Array" ns2:arrayType = "xsd:double[2]">
            <item xsi:type = "xsd:double">54.99</item>
            <item xsi:type = "xsd:double">19.99</item>
         </return>
      </ns1:getPriceListResponse>
   </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

- Structs contain multiple values, but each element is specified with a unique accessor element. For example, consider an item within a product catalog. In this case, the struct might contain a product SKU, product name, description, and price. Here is how such a struct would be represented in a SOAP message −

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd = "http://www.w3.org/2001/XMLSchema">

   <SOAP-ENV:Body>
      <ns1:getProductResponse
         xmlns:ns1 = "urn:examples:productservice"
         SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-
encoding">

         <return xmlns:ns2 = "urn:examples" xsi:type = "ns2:product">
            <name xsi:type = "xsd:string">Red Hat Linux</name>
            <price xsi:type = "xsd:double">54.99</price>
            <description xsi:type = "xsd:string">
               Red Hat Linux Operating System
            </description>
            <SKU xsi:type = "xsd:string">A358185</SKU>
         </return>
      </ns1:getProductResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

### 5.6.8 SOAP-Transport

- SOAP is not tied to any transport protocol. SOAP can be transported via SMTP, FTP, IBM's MQSeries, or Microsoft Message Queuing (MSMQ).
- SOAP specification includes details on HTTP only. HTTP remains the most popular SOAP transport protocol.

### SOAP via HTTP

- Quite logically, SOAP requests are sent via an HTTP request and SOAP responses are returned within the content of the HTTP response. While SOAP requests can be sent via an HTTP GET, the specification includes details on HTTP POST only.

- Additionally, both HTTP requests and responses are required to set their content type to text/xml.
- The SOAP specification mandates that the client must provide a *SOAPAction header,* but the actual value of the SOAPAction header is dependent on the SOAP server implementation.
- For example, to access the AltaVista BabelFish Translation service, hosted by XMethods, you must specify the following as a SOAPAction header.

urn:xmethodsBabelFish#BabelFish

- Even if the server does not require a full SOAPAction header, the client must specify an empty string ("") or a null value. For example −

SOAPAction: ""
SOAPAction:

- Here is a sample request sent via HTTP to the XMethods Babelfish Translation service −

```
POST /perl/soaplite.cgi HTTP/1.0
Host: services.xmethods.com
Content-Type: text/xml; charset = utf-8
Content-Length: 538
SOAPAction: "urn:xmethodsBabelFish#BabelFish"

<?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>
      <ns1:BabelFish
         xmlns:ns1 = "urn:xmethodsBabelFish"
         SOAP-ENV:encodingStyle =
"http://schemas.xmlsoap.org/soap/encoding/">
         <translationmode xsi:type =
"xsd:string">en_fr</translationmode>
         <sourcedata xsi:type = "xsd:string">Hello,
world!</sourcedata>
      </ns1:BabelFish>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Note the content type and the SOAPAction header. Also note that the BabelFish method requires two String parameters. The translation mode en_fr translates from English to French.Here is the response from XMethods −

```
HTTP/1.1 200 OK
Date: Sat, 09 Jun 2001 15:01:55 GMT
Server: Apache/1.3.14 (Unix) tomcat/1.0 PHP/4.0.1pl2
SOAPServer: SOAP::Lite/Perl/0.50
Cache-Control: s-maxage = 60, proxy-revalidate
Content-Length: 539
Content-Type: text/xml

<?xml version = "1.0" encoding = "UTF-8"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENC = "http://schemas.xmlsoap.org/soap/encoding/"
   SOAP-ENV:encodingStyle =
"http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi = "http://www.w3.org/1999/XMLSchema-instance"
   xmlns:SOAP-ENV = "http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd = "http://www.w3.org/1999/XMLSchema">

   <SOAP-ENV:Body>
      <namesp1:BabelFishResponse xmlns:namesp1 =
"urn:xmethodsBabelFish">
         <return xsi:type = "xsd:string">Bonjour, monde!</return>
      </namesp1:BabelFishResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- SOAP responses delivered via HTTP are required to follow the same HTTP status codes. For example, a status code of 200 OK indicates a successful response. A status code of 500 Internal Server Error indicates that there is a server error and that the SOAP response includes a Fault element.

**PART A QUESTIONS & ANSWERS**