**WEB SERVICES**

**What is Web Service?**

Web service is a standardized medium to propagate communication between the client and server applications on the World Wide Web.

Web services provide a common platform that allows multiple applications built on various programming languages to have the ability to communicate with each other

Popular Web Services Protocols are:

**SOAP:**

SOAP is known as the Simple Object Access Protocol.

SOAP was developed as an intermediate language so that applications built on various programming languages could talk quickly to each other and avoid the extreme development effort.

**WSDL:**

WSDL is known as the Web Services Description Language (WSDL).

WSDL is an XML-based file which tells the client application what the web service does and gives all the information required to connect to the web service.

**REST:**

REST stands for REpresentational State Transfer.

REST is used to build Web services that are lightweight, maintainable, and scalable.

**What are Web Services? Architecture, Types, Example**

Modern day business applications use variety of programming platforms to develop web-based applications. Some applications may be developed in Java, others in .Net, while some other in Angular JS, Node.js, etc.

Most often than not, these heterogeneous applications need some sort of communication to happen between them. Since they are built using different development languages, it becomes really difficult to ensure accurate communication between applications.

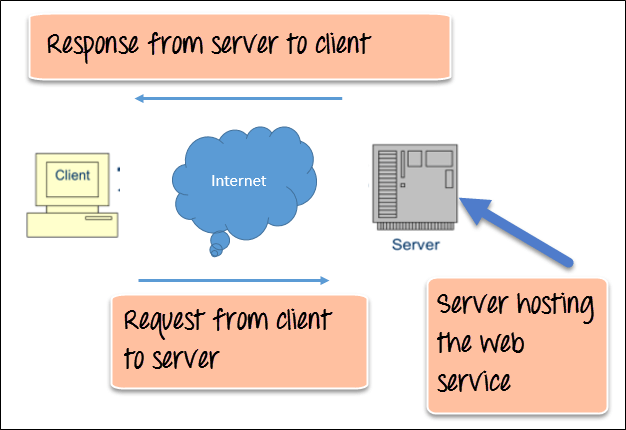
Here is where web services come in. Web services provide a common platform that allows multiple applications built on various programming languages to have the ability to communicate with each other.

**What is Web Service?**

Web service is a standardized medium to propagate communication between the client and server applications on the World Wide Web.

A web service is a software module which is designed to perform a certain set of tasks.

* + The web services can be searched for over the network and can also be invoked accordingly.
  + When invoked the web service would be able to provide functionality to the client which invokes that web service.



Web Service Architecture Diagram

The above diagram shows a very simplistic view of how a web service would actually work. The client would invoke a series of web service calls via requests to a server which would host the actual web service.

These requests are made through what is known as remote procedure calls. Remote Procedure Calls (RPC) are calls made to methods which are hosted by the relevant web service.

The main component of a web service is the data which is transferred between the client and the server, and that is XML. XML (Extensible markup language) is a counterpart to HTML and easy to understand the intermediate language that is understood by many programming languages.

So when applications talk to each other, they actually talk in XML. This provides a common platform for application developed in various programming languages to talk to each other.

Web services use something known as SOAP (Simple Object Access Protocol) for sending the XML data between applications. The data is sent over normal HTTP. The data which is sent from the web service to the application is called a SOAP message. The SOAP message is nothing but an XML document. Since the document is written in XML, the client application calling the web service can be written in any programming language.

**Type of Web Service**

There are mainly two types of web services.

* 1. SOAP web services.
  2. RESTful web services.

In order for a web service to be fully functional, there are certain components that need to be in place. These components need to be present irrespective of whatever development language is used for programming the web service.

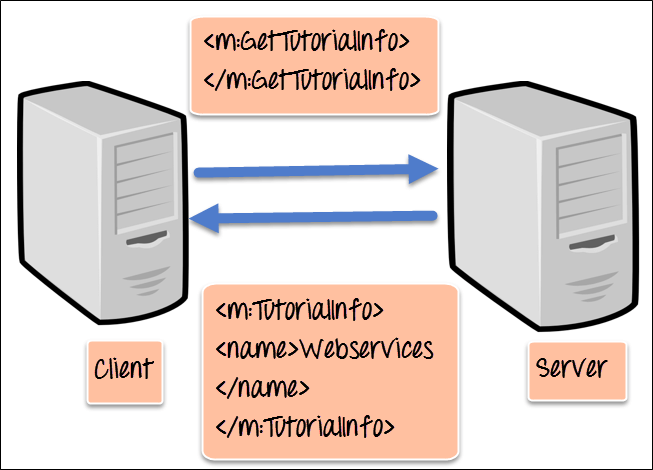
**SOAP (Simple Object Access Protocol)**

SOAP is known as a transport-independent messaging protocol. SOAP is based on transferring XML data as SOAP Messages. Each message has something which is known as an XML document. Only the structure of the XML document follows a specific pattern, but not the content. The best part of Web services and SOAP is that its all sent via HTTP, which is the standard web protocol.

Here is what a SOAP message consists of

* + Each SOAP document needs to have a root element known as the <Envelope> element. The root element is the first element in an XML document.
  + The "envelope" is in turn divided into 2 parts. The first is the header, and the next is the body.
  + The header contains the routing data which is basically the information which tells the XML document to which client it needs to be sent to.
  + The body will contain the actual message.

The diagram below shows a simple example of the communication via SOAP.



**WSDL (Web services description language)**

**A web service cannot be used if it cannot be found**. The client invoking the web service should know where the web service actually resides.

Secondly, the client application needs to know what the web service actually does, so that it can invoke the right web service. This is done with the help of the WSDL, known as the Web services description language. The WSDL file is again an XML-based file which basically tells the client application what the web service does. By using the WSDL document, the client application would be able to understand where the web service is located and how it can be utilized.

**Web Service Example**

An example of a WSDL file is given below.

<definitions>

<message name="TutorialRequest">

<part name="TutorialID" type="xsd:string"/>

</message>

<message name="TutorialResponse">

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

</message>

<portType name="Tutorial\_PortType">

<operation name="Tutorial">

<input message="tns:TutorialRequest"/>

<output message="tns:TutorialResponse"/>

</operation>

</portType>

<binding name="Tutorial\_Binding" type="tns:Tutorial\_PortType">

<soap:binding style="rpc"

transport="http://schemas.xmlsoap.org/soap/http"/>

<operation name="Tutorial">

<soap:operation soapAction="Tutorial"/>

<input>

<soap:body

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

namespace="urn:examples:Tutorialservice"

use="encoded"/>

</input>

<output>

<soap:body

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

namespace="urn:examples:Tutorialservice"

use="encoded"/>

</output>

</operation>

</binding>

</definitions>

The important aspects to note about the above WSDL declaration are as follows;

**Universal Description, Discovery, and Integration (UDDI)**

UDDI is a standard for describing, publishing, and discovering the web services that are provided by a particular service provider. It provides a specification which helps in hosting the information on web services.

Now we discussed in the previous topic about WSDL and how it contains information on what the Web service actually does. But how can a client application locate a WSDL file to understand the various operations offered by a web service? So UDDI is the answer to this and provides a repository on which WSDL files can be hosted. So the client application will have complete access to the UDDI, which acts as a database containing all the WSDL files.

**Just as a telephone directory has the name, address and telephone number of a particular person, the same way the UDDI registry will have the relevant information for the web service**. So that a client application knows, where it can be found.

**Web Services Advantages**

We already understand why web services came about in the first place, which was to provide a platform which could allow different applications to talk to each other.

But let's look at some other advantages of why it is important to use web services.

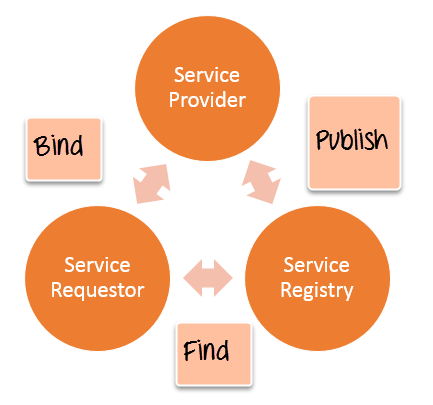
* 1. **Exposing Business Functionality on the network** - A web service is a unit of managed code that provides some sort of functionality to client applications or end users. This functionality can be invoked over the HTTP protocol which means that it can also be invoked over the internet. Nowadays all applications are on the internet which makes the purpose of Web services more useful. That means the web service can be anywhere on the internet and provide the necessary functionality as required.
  2. **Interoperability amongst applications** - Web services allow various applications to talk to each other and share data and services among themselves. All types of applications can talk to each other. So instead of writing specific code which can only be understood by specific applications, you can now write generic code that can be understood by all applications
  3. **A Standardized Protocol which everybody understands** - Web services use standardized industry protocol for the communication. All the four layers (Service Transport, XML Messaging, Service Description, and Service Discovery layers) uses well-defined protocols in the web services protocol stack.
  4. **Reduction in cost of communication** - Web services use SOAP over HTTP protocol, so you can use your existing low-cost internet for implementing web services.

**Web service Architecture**

Every framework needs some sort of architecture to make sure the entire framework works as desired. Similarly, in web services, there is an architecture which consists of three distinct roles as given below

* 1. **Provider** - The provider creates the web service and makes it available to client application who want to use it.
  2. **Requestor** - A requestor is nothing but the client application that needs to contact a web service. The client application can be a .Net, Java, or any other language based application which looks for some sort of functionality via a web service.
  3. **Broker** - The broker is nothing but the application which provides access to the UDDI. The UDDI, as discussed in the earlier topic enables the client application to locate the web service.

The diagram below showcases how the Service provider, the Service requestor and Service registry interact with each other.



* 1. **Publish** - A provider informs the broker (service registry) about the existence of the web service by using the broker's publish interface to make the service accessible to clients
  2. **Find** - The requestor consults the broker to locate a published web service
  3. **Bind** - With the information it gained from the broker(service registry) about the web service, the requestor is able to bind, or invoke, the web service.

**Web service Characteristics**

Web services have the following special behavioral characteristics:

* 1. **They are XML-Based** - Web Services uses XML to represent the data at the representation and data transportation layers. Using XML eliminates any networking, operating system, or platform sort of dependency since XML is the common language understood by all.
  2. **Loosely Coupled** – Loosely coupled means that the client and the web service are not bound to each other, which means that even if the web service changes over time, it should not change the way the client calls the web service. Adopting a loosely coupled architecture tends to make software systems more manageable and allows simpler integration between different systems.
  3. **Synchronous or Asynchronous** **functionality**- Synchronicity refers to the binding of the client to the execution of the service. In synchronous operations, the client will actually wait for the web service to complete an operation. An example of this is probably a scenario wherein a database read and write operation are being performed. If data is read from one database and subsequently written to another, then the operations have to be done in a sequential manner. Asynchronous operations allow a client to invoke a service and then execute other functions in parallel. This is one of the common and probably the most preferred techniques for ensuring that other services are not stopped when a particular operation is being carried out.
  4. **Ability to support Remote Procedure Calls (RPCs)** - Web services enable clients to invoke procedures, functions, and methods on remote objects using an XML-based protocol. Remote procedures expose input and output parameters that a web service must support.
  5. **Supports Document Exchange** - One of the key benefits of XML is its generic way of representing not only data but also complex documents. These documents can be as simple as representing a current address, or they can be as complex as representing an entire book.

**SOAP Web Services Tutorial: Simple Object Access Protocol EXAMPLE**

**What is SOAP?**

SOAP is an XML-based protocol for accessing web services over HTTP. It has some specification which could be used across all applications.

SOAP is known as the Simple Object Access Protocol, but in later times was just shortened to SOAP v1.2. SOAP is a protocol or in other words is a definition of how web services talk to each other or talk to client applications that invoke them.

SOAP was developed as an intermediate language so that applications built on various programming languages could talk easily to each other and avoid the extreme development effort.

**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**

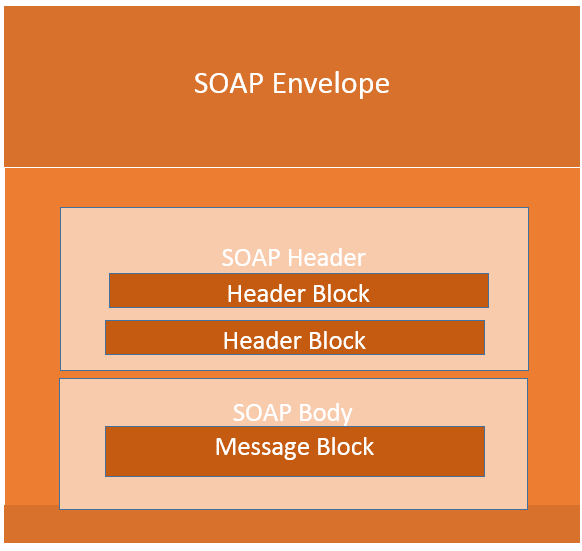
SOAP is the protocol used for data interchange between applications. Below are some of the reasons as to why SOAP is used.

* 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 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.



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**

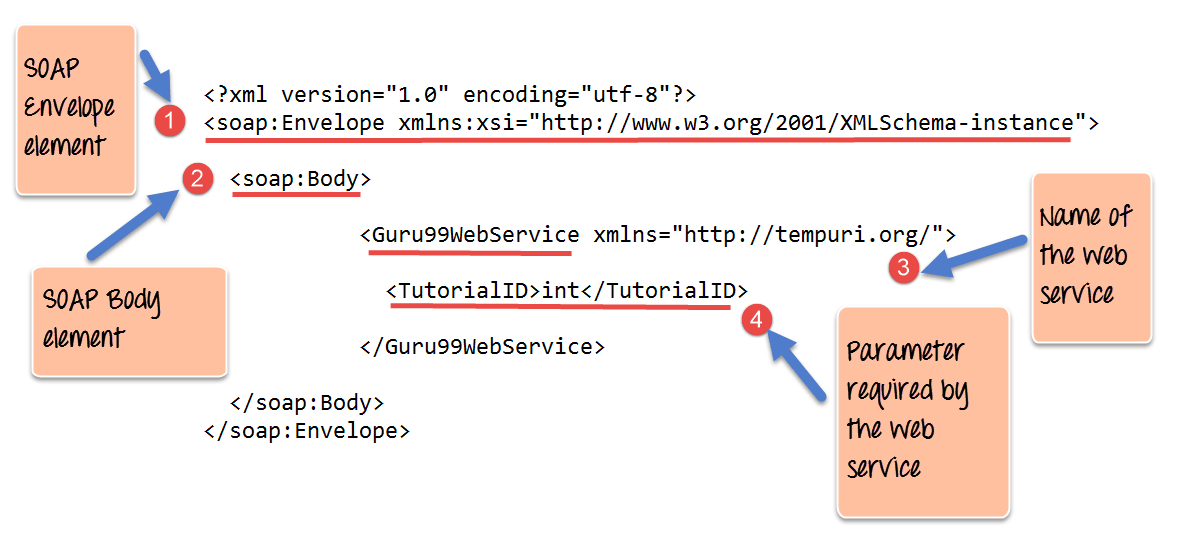
One thing to note is that SOAP messages are normally auto-generated by the web service when it is called.

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)"

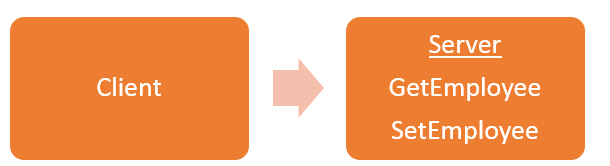
**SOAP Communication Model.**

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.

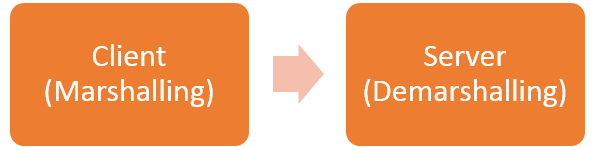
In the normal RPC style communication, the client would just call the methods in its request and send the required parameters to the server, and the server would then send the desired response.

[](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.

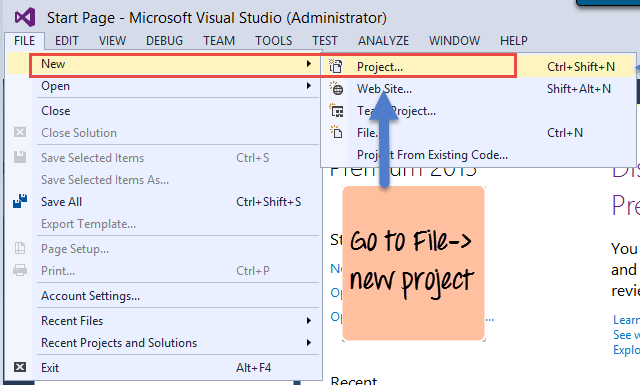
This web service will be hosted in an Asp.Net web application. We will then invoke the web service and see the result that is returned by the web service.

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.

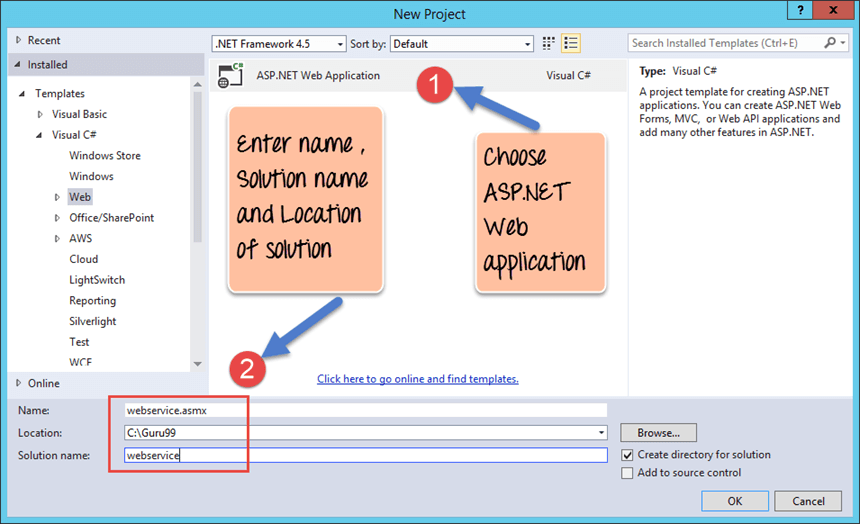
**Step 1)** The first step is to create an empty ASP.Net Web application. From Visual Studio 2013, click on the menu option File->New project.

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

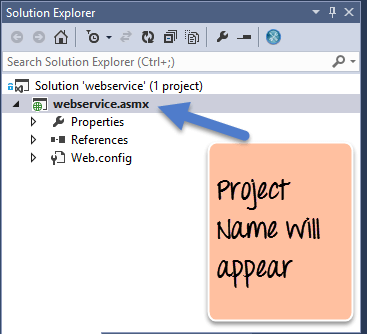
Once you click on the New Project option, Visual Studio will then give you another dialog box for choosing the type of project and to give the necessary details of the project. This is explained in the next step.

**Step 2)** In this step,

1. Ensure to first choose the C# 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)

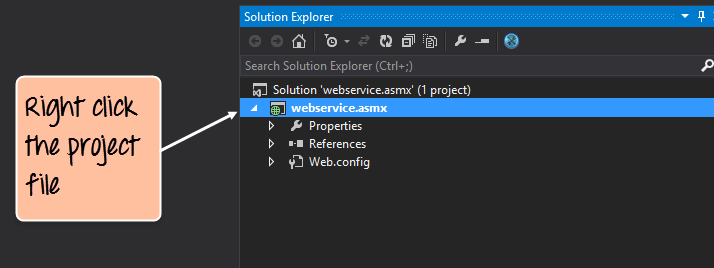
Once done you will see the project file created in your solution explorer in Visual Studio 2013.

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

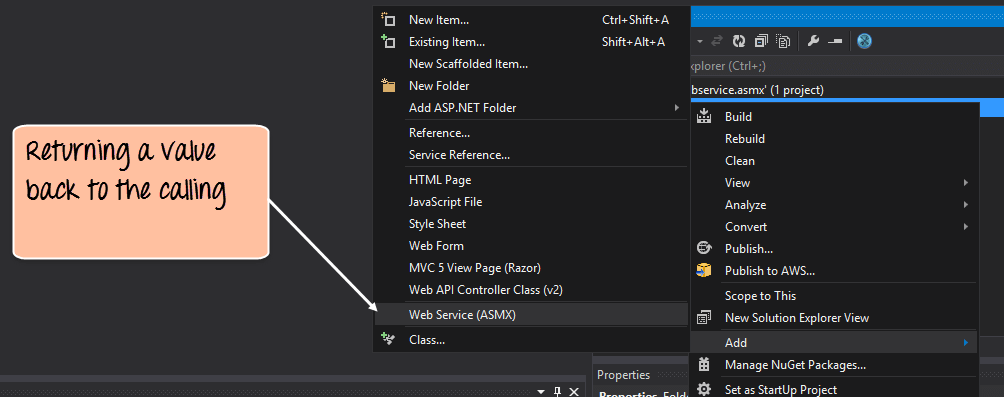
**Step 3)** In this step,

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

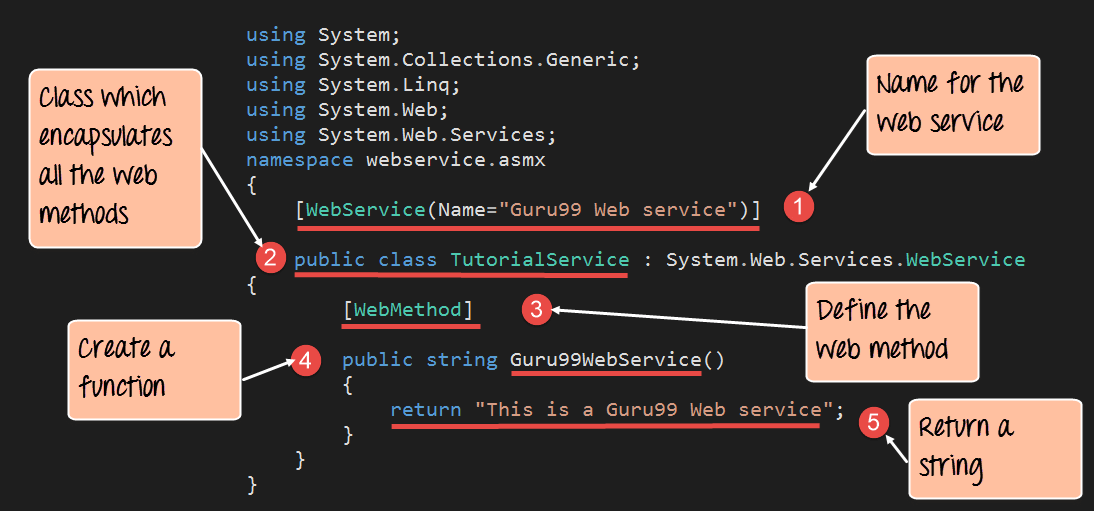
1. First Right-click on the project file as shown below

[](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.

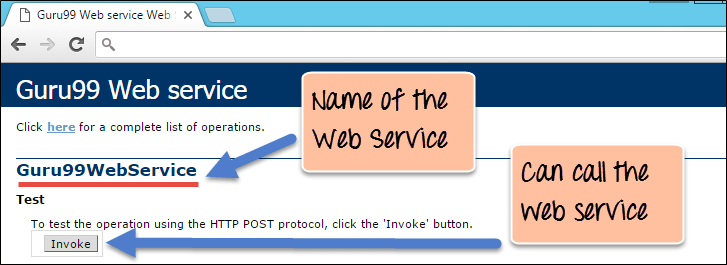
[](https://www.guru99.com/images/3-2016/032316_0711_SOAPSimpleO10.png)

**Code Explanation:**

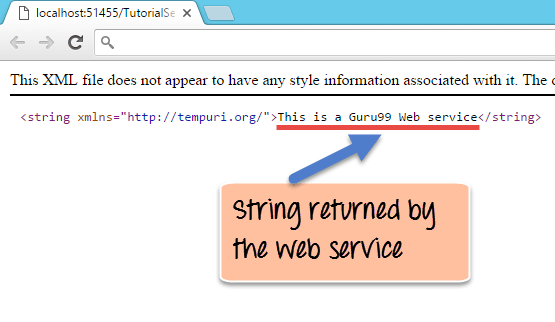
1. This line of code provides a name for your web service file. This is an important step because it gives way for the client application to call the web service via the name of the web service.
2. Normally a class file is used to encapsulate the functionality of a web service. So the class file will 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)

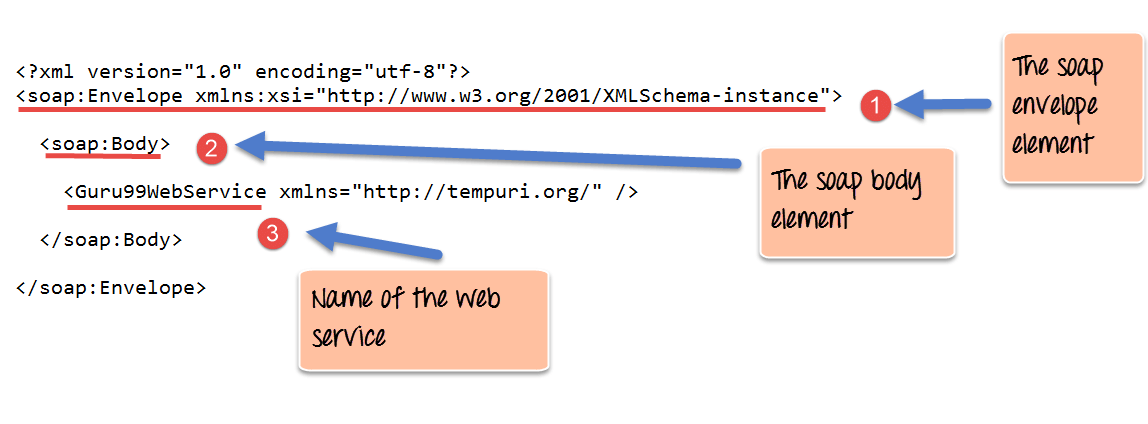
* The output clearly shows that the name of our web service is "Guru99 Web Service" which is the result of giving a name for our web service.
* We can also see that we can to invoke the web service. If we click the Invoke button, we will get the below response in the web browser.

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

The above output,

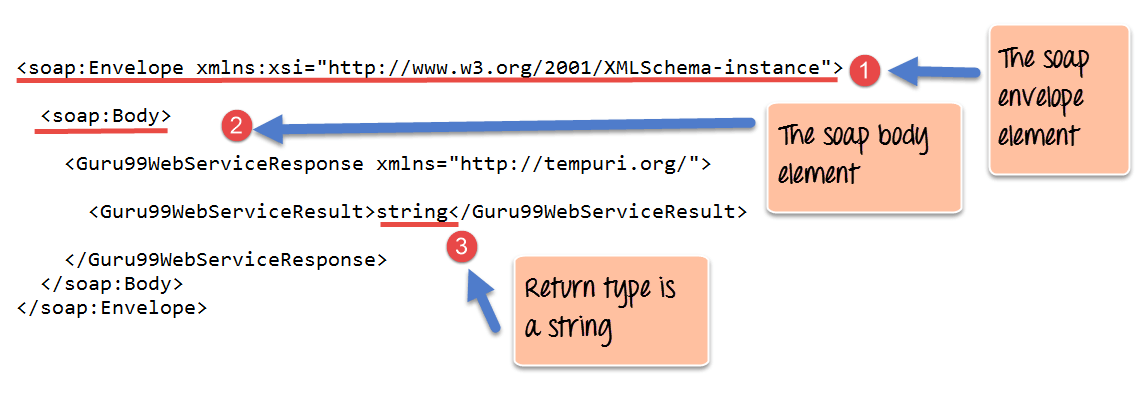
* It clearly shows that by invoking the web method, the string "This is a Gu99 Web service" is returned.
* Visual Studio also allows you to view the SOAP message request and response which is generated when the above web service is called.

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.

**What is 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).

In this tutorial, we are going to focus on the last point which is the most important part of web services, and that is the WSDL or the Web services description language.

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.

**Structure of a WSDL Document**

A WSDL document is used to describe a web service. This description is required, so that client applications are able to understand what the web service actually does.

* The WSDL file contains the location of the web service and
* The methods which are exposed by the web service.

The WSDL file itself can look very complex to any user, but it contains all the necessary information that any client application would require to use the relevant web service.

Below is the general structure of a WSDL file

* Definition
* TargetNamespace
* DataTypes
* Messages
* Porttype
* Bindings
* service

One key thing to note here is that definition of messages, which is what is passed by the SOAP protocol is actually defined in the WSDL document.

The WSDL document actually tells a client application what are the types of SOAP messages which are sent and accepted by the Web service.

In other words, the WSDL is just like a postcard which has the address of a particular location. The address provides the details of the person who delivered the postcard. Hence, in the same way, the WSDL file is the postcard, which has the address of the web service which can deliver all the functionality that the client wants.

<!-- WSDL definition structure -->

<definitions

name="Guru99Service"

targetNamespace=http://example.org/math/

xmlns=http://schemas.xmlsoap.org/wsdl/>

<!-- abstract definitions -->

<types> ...

<message> ...

<portType> ...

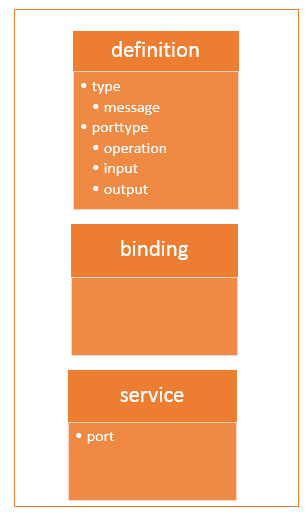
<!-- concrete definitions -->

<binding> ...

<service> ...

</definition>

Below is a diagram of the structure of a WSDL file

[](https://www.guru99.com/images/3-2016/032316_0742_WSDLWebserv1.png)

**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.

1. 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.
2. 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.
3. 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.
4. 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.

**Why WSDL**

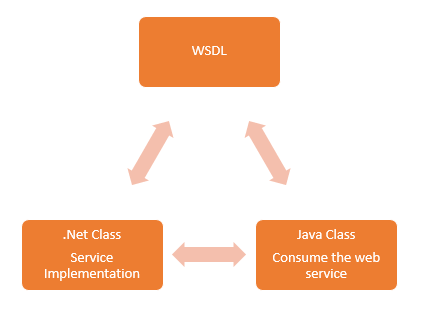
A web service is an important component in building modern day web applications. Their main purpose is to allow multiple applications built on various programming languages to talk to each other. For instance, we can have a .Net web application talks to a[Java](https://www.guru99.com/java-tutorial.html)application via a Web service.

A web service has the following key features

* It is built using the XML programming language. Almost all modern day technologies such as .Net and Java have corresponding commands that have the ability to work with XML. Hence, XML was taken as the most appropriate language for building web services.
* Web services communicate over HTTP. HTTP is a protocol used by all web-based applications. Hence, it just made sense to ensure that Web services also had the ability to work over the HTTP protocol.
* Web services conform to a particular language specification. This specification is set by the W3C, which is the governing body for all web standards.
* Web services have a description language known as WSDL, which is used to describe the web service.

The WSDL file is written in plain old XML. The reason that it is in XML is so that the file can be read by any programming language.

So if the client application was written in .Net, it would understand the XML file. Similarly, if the client application was written in the Java programming language then also it would be able to interpret the WSDL file.



The WSDL file is what binds everything together. From the above diagram, you can see that you can create a web service in the .Net language.

So this is where the service gets implemented. If you did not have the WSDL file and wanted a Java class to consume the web service, you would need a lot of coding effort to achieve this.

But now with the WSDL file which is in XML, which can be understood by any programming language, you can now easily have a Java class consume the .Net web service. Hence, the amount of coding effort is greatly reduced.

**WSDL Message Part**

The WSDL consists of a section called "messages" which is denoted by the **<message>**element.

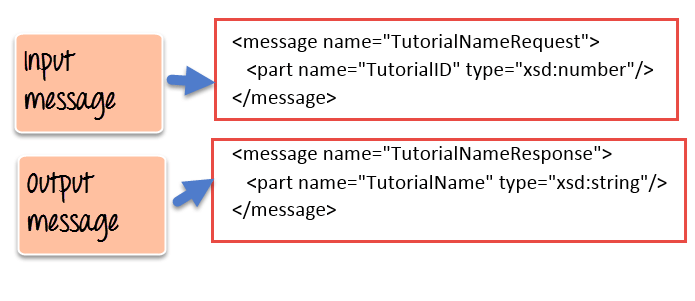
This element is basically used to describe the data that gets exchanged between the web service and the client application.

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.

[](https://www.guru99.com/images/3-2016/032316_0742_WSDLWebserv3.png)

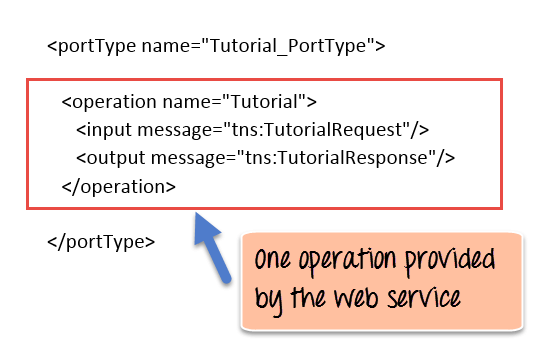
1. As we can see the web service has 2 messages, one for the input and the other for the output.
2. The input message is known as TutorialNameRequest which has one parameter called TutorialID. This parameter is of the type number which is specified by the xsd:number type
3. The output message is known as TutorialNameResponse which has one parameter called TutorialName. This parameter is of the type string which is specified by the xsd:string type

**Port Type Binding**

Ports are used in WSDL to define one complete operation which is offered by the web service.

In the previous topic, we saw that our web service provided 2 messages, one for the input called "TutorialNameRequest" and the other for the output called "TutorialNameResponse." Together the input and output message form is known as one complete operation.

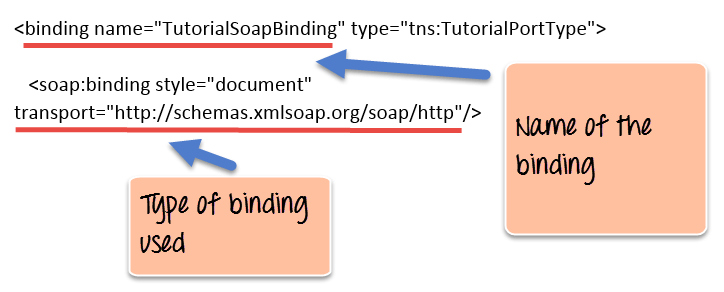
WSDL provides an element called **<portType>** which is used to define the operations provided by the Web service.

[](https://www.guru99.com/images/3-2016/032316_0742_WSDLWebserv4.png)

So in our above example we can note the following:

1. The name of the port Type which encapsulates the operation is given as "Tutorial\_PortType."
2. The operation itself is given a name of "Tutorial". So our operation basically provides a TutorialName if the TutorialID is given as an input parameter.
3. Next is our 2 messages, one for the input and the other for the output which forms our operation

In addition to the **<portType>** element, there is also the **<binding>** element which is used to define how the messages will be transferred.

[](https://www.guru99.com/images/3-2016/032316_0742_WSDLWebserv5.png)

1. The above example shows that the binding consists of a binding name which in our case is given as "TutorialSoapBinding". Binding in simple terms is the information which the client application uses to actually bind itself to the web service. Once it is actually bound to the web service, it then has the ability to call the various operations that are exposed by the web service.
2. The transport layer is given as http:// which means that the messages which will transfer over the HTTP protocol.

**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>

The above WSDL file looks very intimidating to any user, we will cover the different parts in detail in the subsequent tutorials, but for now, let's have a summary look at what each section of the WSDL file actually does.

**Publishing the Web Service Example**

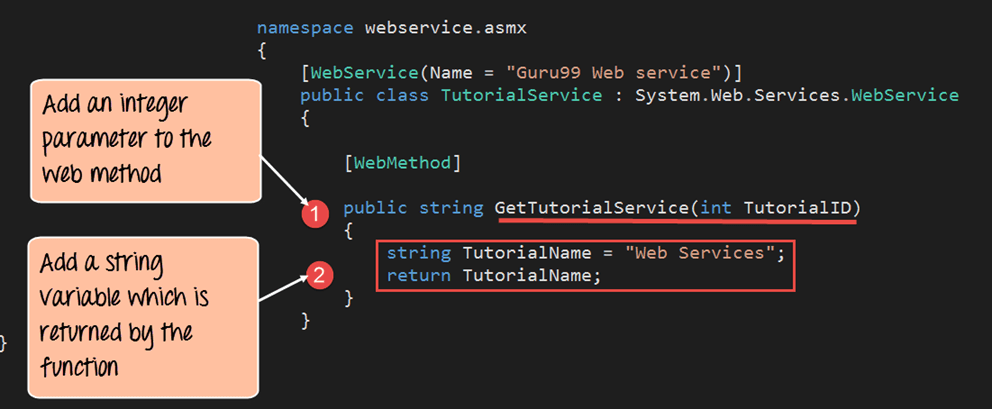
Now let's look at an example of how we can publish a web service and consume it by using Visual Studio.

In this example, we will create a web service with one WebMethod. This method will accept an Integer parameter called "TutorialID." The Web method will then return a string called "Web Services."

We will then create a console based application, which will consume this web service and call our web method accordingly.

Let's look at the steps required to carry out this example.

**Step 1)** The first step is to create your web service. The detailed steps of how the[Asp.Net](https://www.guru99.com/asp-net-tutorial.html)web project and a web service is created has been explained [here;](https://www.guru99.com/soap-simple-object-access-protocol.html) Please follow the same steps to create the project and web service accordingly. The key part is to enter the below code in the Web services file.

[](https://www.guru99.com/images/3-2016/032316_0742_WSDLWebserv6.png)

namespace webservic asmx

{

[WebService(Name = "Guru99 Web service")]

public class TutorialService : System.Web.Services.WebService

{

[WebMethod]

public string GetTutorialService(int TutoriallD)

{

string TutorialName = "Web Services";

return TutorialName;

}

}

}

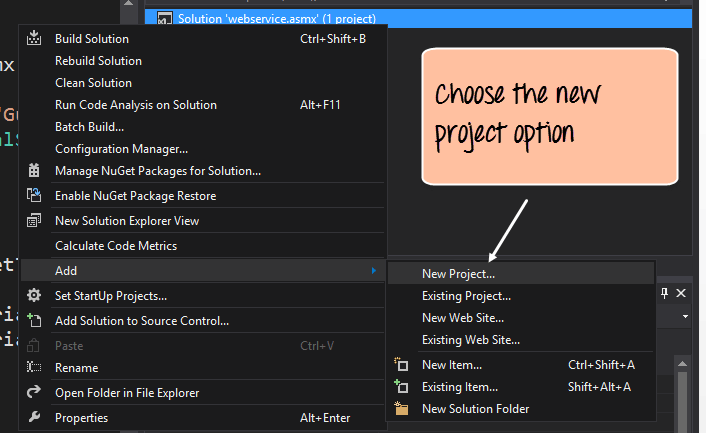
**Code Explanation:**

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.

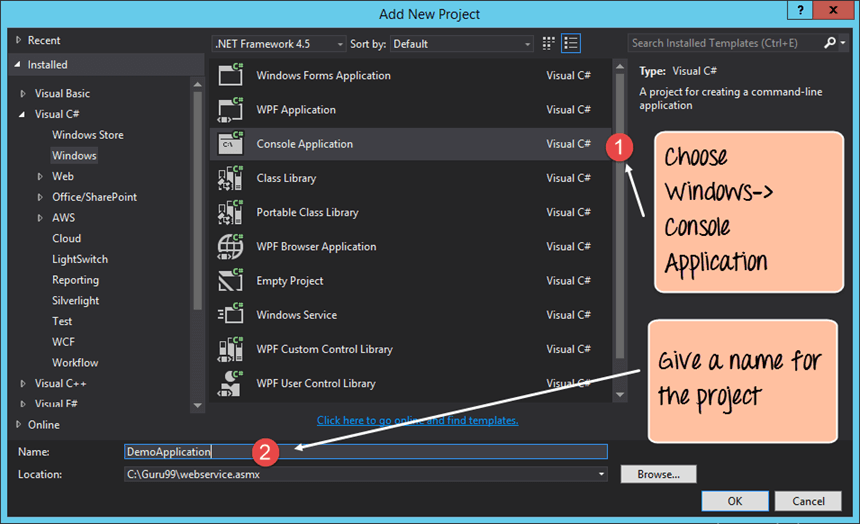
Let's create a simple console application which will call this web service, invoke the "Guru99WebService" and then display the output of the web method in the console log screen. Follow the below steps to create a console application.

Right-click the Visual Studio solution file and choose the option Add->New project

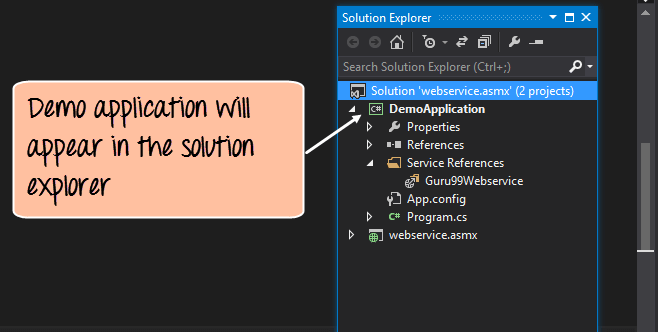
[](https://www.guru99.com/images/3-2016/032316_0742_WSDLWebserv7.png)

**Step3)** In this step,

1. Ensure to first choose the Visual[C#](https://www.guru99.com/c-tutorial.html)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."

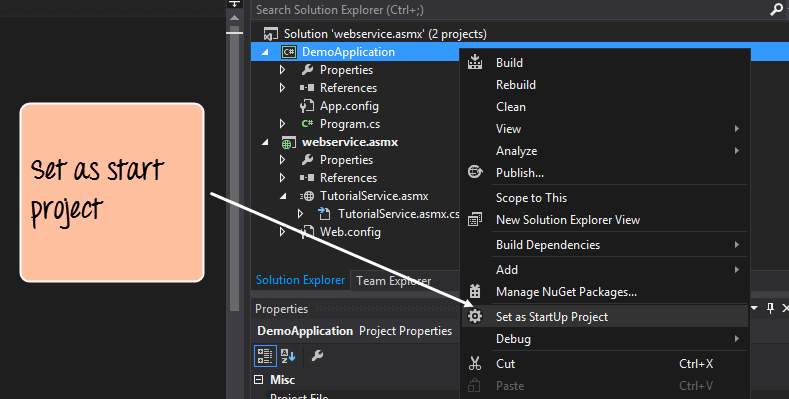
[](https://www.guru99.com/images/3-2016/032316_0742_WSDLWebserv8.png)

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.

[](https://www.guru99.com/images/3-2016/032316_0742_WSDLWebserv9.png)

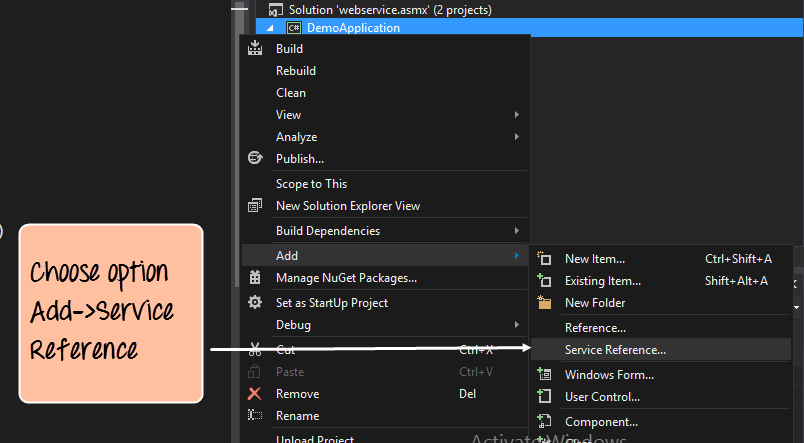
**Step 4)** In this step, you be setting the DemoApplication Console application as the startup project. This is done to ensure that this application launches first when the entire Visual Studio project is run. This Console application will, in turn, call the web service which will be automatically launched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

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

[](https://www.guru99.com/images/3-2016/032316_0742_WSDLWebserv10.png)

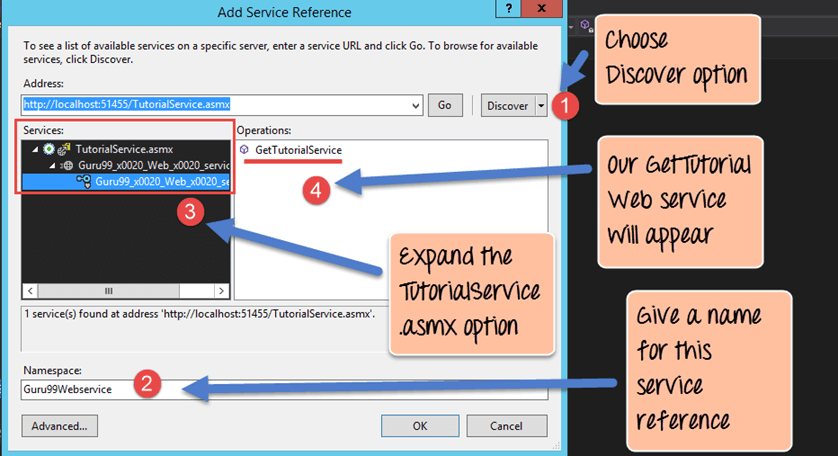
**Step 5)**The next step is to add the service reference of our "Guru99Webservice" to our console application. This is done so that the DemoApplication can reference the web service and all of the web methods in the web service.

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

[](https://www.guru99.com/images/3-2016/032316_0742_WSDLWebserv11.png)

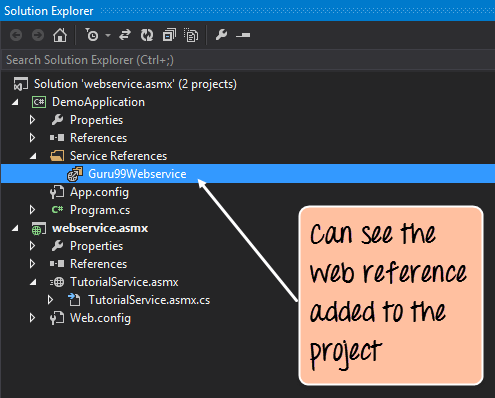
**Step 6)**In this step, we will provide the different values which are required to add our service reference

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"

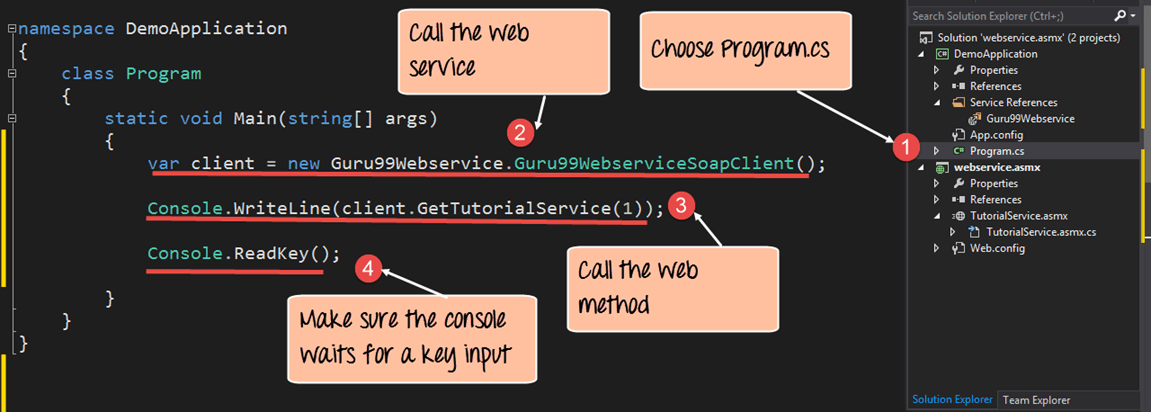
[](https://www.guru99.com/images/3-2016/032316_0742_WSDLWebserv12.png)

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.

[](https://www.guru99.com/images/3-2016/032316_0742_WSDLWebserv13.png)

**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

[](https://www.guru99.com/images/3-2016/032316_0742_WSDLWebserv14.png)

namespace DemoApplication

{

class Program

{

static void Main(string[ ] args)

{

var client = new Guru99Webservice.Guru99WebserviceSoapClient();

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

Console.ReadKey();

}

}

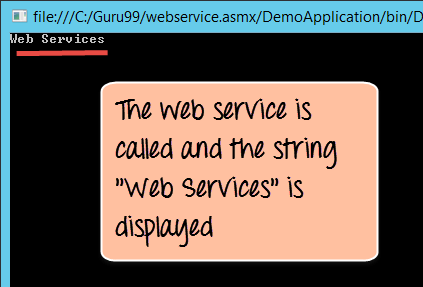
}

**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.

[](https://www.guru99.com/images/3-2016/032316_0742_WSDLWebserv15.png)

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.

**Summary**

* A WSDL document is a document that is used to describe a web service. This is key for any client application to know where the web service is located. It also allows the client application to understand the methods available in the web service.
* The WSDL file makes it very easy for the web service to be implemented in one programming language and called from a different programming language.
* The WSDL document normally consists of a message. For each web method, there are 2 messages, one is for the input, and the other is for the output. Together they form an operation.
* WSDL files normally get created in the editor which is used for the corresponding programming language.
* We have seen how we can consume a web service in Visual Studio. This can be done by creating another project which is a console application. Then by adding a service reference, we are then able to access the web methods in our web service.