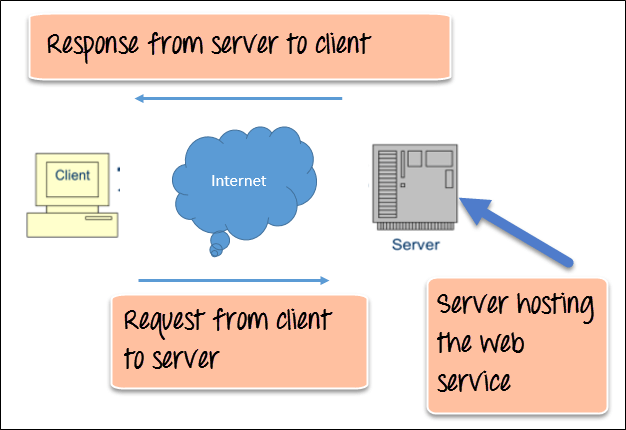
**What is Web Service?**

**Web service** is a standardized medium to propagate communication between the client and server applications on the WWW (World Wide Web). A web service is a software module that is designed to perform a certain set of tasks.

* Web services in cloud computing can be searched for over the network and can also be invoked accordingly.
* When invoked, the web service would be able to provide the functionality to the client, which invokes that web service.

## How WebServices Work?

How Web Services Work?

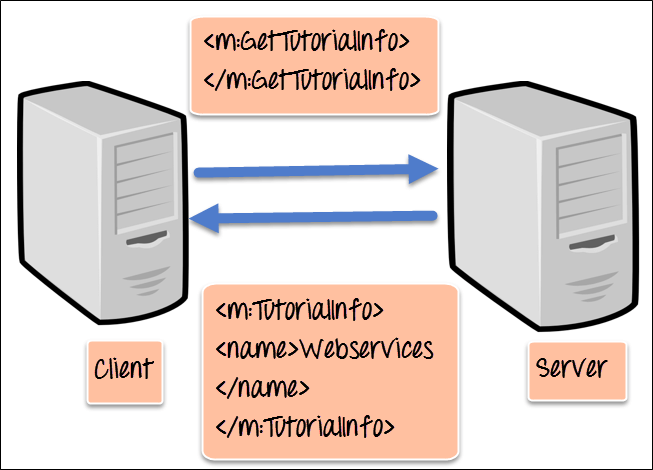
The above diagram shows a very simplistic view of how a web service would actually work. The client would invoke a series of web service calls via requests to a server which would host the actual web service.

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

SOAP Protocol

We will discuss SOAP in detail in this [tutorial](https://www.guru99.com/soap-simple-object-access-protocol.html).

**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

A Web services 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 examples of web services are as follows:

1. **<message>** – The message parameter in the WSDL definition is used to define the different data elements for each operation performed by the web service. So in the web services examples above, we have 2 messages which can be exchanged between the web service and the client application, one is the “TutorialRequest”, and the other is the “TutorialResponse” operation. The TutorialRequest contains an element called “TutorialID” which is of the type string. Similarly, the TutorialResponse operation contains an element called “TutorialName” which is also a type string.
2. **<portType>** – This actually describes the operation which can be performed by the web service, which in our case is called Tutorial. This operation can take 2 messages; one is an input message, and the other is the output message.
3. **<binding>** – This element contains the protocol which is used. So in our case, we are defining it to use http (**http://schemas.xmlsoap.org/soap/http**). We also specify other details for the body of the operation, like the namespace and whether the message should be encoded.

We will discuss “WDSL” in detail in this [tutorial](https://www.guru99.com/wsdl-web-services-description-language.html).

**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 the list of web services advantages for why it is important to use web services.

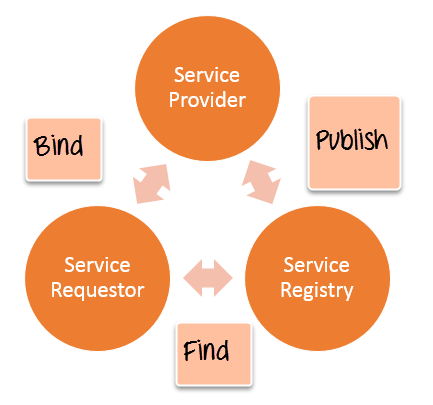
1. **Exposing Business Functionality on the network** – A web service is a unit of managed code that provides some sort of functionality to client applications or end users. This functionality can be invoked over the HTTP protocol which means that it can also be invoked over the internet. Nowadays all 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 Services Architecture**

Every framework needs some sort of architecture to make sure the entire framework works as desired, similarly, in web services. The **Web Services Architecture** 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.

Web Services Architecture

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.

**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.

In this SOAP Web services tutorial, you will learn-

* [SOAP Introduction](https://www.guru99.com/soap-simple-object-access-protocol.html#1)
* [Advantages of SOAP](https://www.guru99.com/soap-simple-object-access-protocol.html#2)
* [SOAP Building blocks](https://www.guru99.com/soap-simple-object-access-protocol.html#3)
* [SOAP Message Structure](https://www.guru99.com/soap-simple-object-access-protocol.html#4)
* [SOAP Envelope Element](https://www.guru99.com/soap-simple-object-access-protocol.html#5)
* [SOAP Communication Model](https://www.guru99.com/soap-simple-object-access-protocol.html#6)
* [Practical SOAP Example](https://www.guru99.com/soap-simple-object-access-protocol.html#7)

**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](https://www.guru99.com/php-tutorials.html).

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 software 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

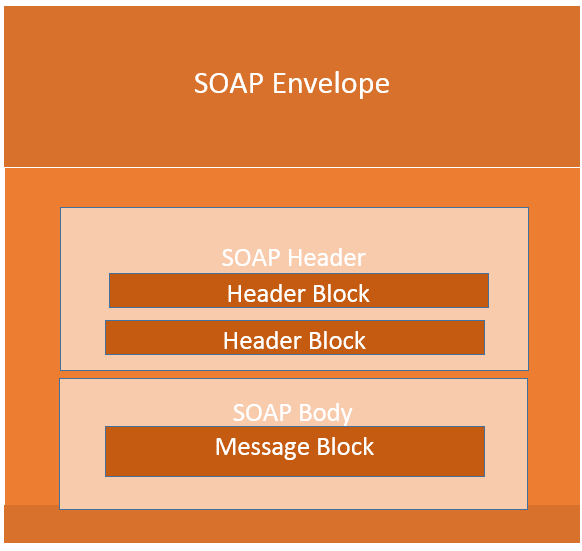
SOAP is the protocol used for data interchange between applications. Below are some of the reasons as to why SOAP is used.

* When developing SOAP based 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 programming is based on the XML language, which itself is a light weight data interchange language, hence SOAP as a protocol that also falls in the same category.
* SOAP is designed to be platform independent and is also designed to be operating system independent. So the SOAP protocol can work any programming language based applications on both Windows and [Linux](https://www.guru99.com/unix-linux-tutorial.html) platform.
* It works on the HTTP protocol –SOAP works on the HTTP protocol, which is the default protocol used by all web applications. Hence, there is no sort of customization which is required to run the web services built on the SOAP protocol to work on the World Wide Web.

**SOAP Building Blocks**

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

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

SOAP Message Building Blocks

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 SOAP service 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 SOAP web service 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**

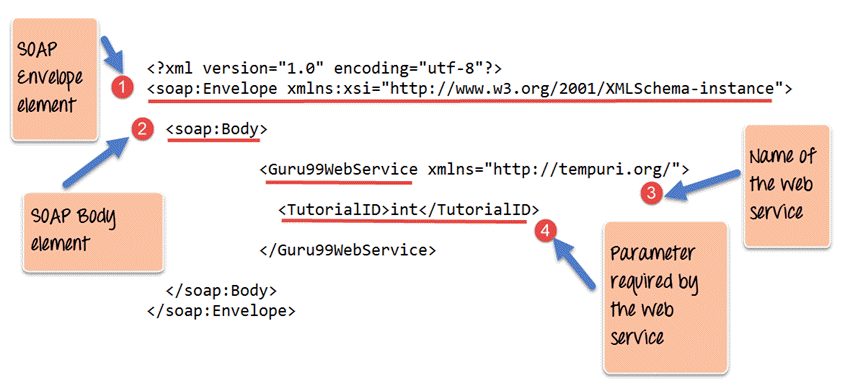
One thing to note is that SOAP messages are normally auto-generated by the web service when it is called.

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 of this SOAP tutorial, 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.

SOAP Message Structure

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 SOAP API 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.