WSDL 2.0 - Web Service Description Language

WSDL is short for Web Service Description Language. WSDL is used to describe the interface of a web service. If you do not know the interface of a web service, how could you call it?

This WSDL tutorial is an introduction to WSDL 2.0. It will not cover each and every little detail of WSDL 2.0, but focus on the primary details. Parts of this tutorial will be reworked over time, so if you have any corrections etc., let me know. See the bottom of the page for how to contact me.

**WSDL is Written in XML**

Like SOAP, WSDL is written in XML. This makes WSDL documents platform independent. Most programming languages and platforms have XML parsing tools these days, so no matter what language or platform you are using, you should be able to parse WSDL files.

**WSDL is Machine Readable**

WSDL is machine readable, meaning a program could read a WSDL file and from that deduce how to call a given web service. You can also generate code for clients from a WSDL file. You can even generate stubs for your own web services from a WSDL file.

**WSDL Tutorial - Table of Contents**

In this WSDL tutorial I will describe how to write your own WSDL files. First I will give you an overview of WSDL in the first text. Then I will describe each element, each in it's own text.

**WSDL 2.0**

This WSDL tutorial focuses on WSDL 2.0 . The major new thing in WSDL 2.0 is the interface element, which describes a web service interface in a more intuitive way than the previous portType, operation and message elements did. The interface element groups operation and messageelements naturally together, instead of listing them separately, as was the practice in earlier versions of WSDL.

A WSDL file is an XML file describing the interface of a web service. A WSDL 2.0 file contains the following elements:

|  |  |
| --- | --- |
| description | The description element is the root element of the WSDL 2.0 file. All other WSDL elements are nested inside this element. |
| types | The types element contains a specification of the data types exchanged between the client and the web service. By default these data types are described using XML Schema. |
| interface | The interface element describes what operations the web service has, and what messages are exchanged for each operation (input / output). It also describes possible fault messages. |
| binding | The binding element describes **how** the web service is accessed over the network. Typically the binding element binds the web service to the HTTP protocol. |
| service | The service element describes **where** the web service can be accessed on the network. Typically the service element contains a URL to the service. |
| documentation | The documentation element is optional and may contain a humanly readable description of the web service. |
| import | The import element is optional and may be used to import XML Schemas or other WSDL files. |

Each of these elements are explained in more detail in their own texts. See the top corner of this page for links to these texts.

Here is an outline of the required XML elements in a WSDL file:

<description>

<types>

</types>

<interface>

</interface>

<binding>

</binding>

<service>

</service>

</description>

**WSDL Compared to Java Interfaces**

Since WSDL describes the interface of a web service, I thought it might be useful for you to see how the XML elements in the WSDL match the declaration of a Java interface. That is what the diagram below shows. The orange boxes represent elements in the WSDL file.

|  |
| --- |
| WSDL compared to a Java interface. |
| **WSDL compared to a Java interface. The orange boxes are WSDL elements.** |

As you can see, the Java classes (Input and Output) would be defined in the types element in WSDL. So would the exception. The method in the Java interface would be declared in an operation element inside the WSDL interface element. You'll see more examples on that in the [**interface**](http://tutorials.jenkov.com/wsdl/interface.html) text.

**Full WSDL Example**

Here is a full WSDL example:

<?xml version="1.0" encoding="utf-8" ?>

<**description**

xmlns= "http://www.w3.org/ns/wsdl"

targetNamespace= "http://jenkov.com/MyService"

xmlns:tns= "http://jenkov.com/MyService"

xmlns:stns = "http://jenkov.com/MyService/schema"

xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"

xmlns:soap= "http://www.w3.org/2003/05/soap-envelope"

xmlns:wsdlx= "http://www.w3.org/ns/wsdl-extensions" >

<**documentation**>

This is the web service documentation.

</**documentation**>

<**types**>

<xs:schema

xmlns:xs= "http://www.w3.org/2001/XMLSchema"

targetNamespace= "http://jenkov.com/MyService/schema"

xmlns= "http://jenkov.com/MyService/schema">

<xs:element name="latestTutorialRequest"

type="typeLatestTutorialRequest"/>

<xs:complexType name="typeLatestTutorialRequest">

<xs:sequence>

<xs:element name="date" type="xs:date"/>

</xs:sequence>

</xs:complexType>

<xs:element name="latestTutorialResponse" type="xs:string"/>

<xs:element name="invalidDateError" type="xs:string"/>

</xs:schema>

</**types**>

<**interface** name = "latestTutorialInterface" >

<fault name = "invalidDateFault" element = "stns:invalidDateError"/>

<operation name="latestTutorialOperation"

pattern="http://www.w3.org/ns/wsdl/in-out"

style="http://www.w3.org/ns/wsdl/style/iri"

wsdlx:safe = "true">

<input messageLabel="In" element="stns:latestTutorialRequest" />

<output messageLabel="Out" element="stns:latestTutorialResponse" />

<outfault messageLabel="Out" ref ="tns:invalidDateFault" />

</operation>

</**interface**>

<**binding** name="latestTutorialSOAPBinding"

interface="tns:latestTutorialInterface"

type="http://www.w3.org/ns/wsdl/soap"

wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">

<fault ref="tns:invalidDateFault" wsoap:code="soap:Sender"/>

<operation ref="tns:latestTutorialOperation"

wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>

</**binding**>

<**service**

name ="latestTutorialService"

interface="tns:latestTutorialInterface">

<endpoint name ="latestTutorialEndpoint"

binding ="tns:latestTutorialSOAPBinding"

address ="http://jenkov.com/latestTutorial"/>

</**service**>

</**description**>

The WSDL description element is the root element of a WSDL 2.0 file. The description element begin tag usually contains a set of name space declarations which are used throughout the WSDL file.

Here is an example description element:

<?xml version="1.0" encoding="utf-8" ?>

<**description**

xmlns= "http://www.w3.org/ns/wsdl"

targetNamespace= "http://jenkov.com/MyService"

xmlns:tns= "http://jenkov.com/MyService"

xmlns:stns = "http://jenkov.com/MyService/schema"

xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"

xmlns:soap= "http://www.w3.org/2003/05/soap-envelope"

xmlns:wsdlx= "http://www.w3.org/ns/wsdl-extensions" >

</**description**>

Each of these name spaces are explained below.

**xmlns**

The xmlns attribute sets the default name space of the description element. The default name space is thus applied to all elements inside the description element, which does not explicitly declare another name space for themselves.

The default name space is set to a standard value for WSDL files: http://www.w3.org/ns/wsdl

**targetNameSpace**

This attribute contains the name space of your web service. You can choose this name space freely, but there is a convention saying that the URI should point to the WSDL of the service.

**xmlns:tns**

This name space should be set to the same URI as the targetNameSpaceattribute. That way you can refer to the target name space via this name space prefix (tns).

**xmlns:stns**

This name space attribute declares the Schema Target Name Space URI. In other words, it should point to the URI for the XML schema's name space, of the schema you declare for your web service types, in the types element.

**xmlns:wsoap**

Points to the WSDL SOAP URI. Used in the bindings element of the WSDL.

**xmlns:soap**

Points to the SOAP URI of the SOAP version the web service described by the WSDL is using.

**xmlns:wsdlx**

Points to the WSDL Extensions URI.

**WSDL 2.0 - types**

* [XML Schema: Top Level Elements Only](http://tutorials.jenkov.com/wsdl/types.html#xml-schema-top-level-elements-only)
* [The Schema Name Space](http://tutorials.jenkov.com/wsdl/types.html#the-schema-name-space)

The WSDL types element describes the data types used by your web service. Most often a web service will have an input type, an output type, and perhaps a fault type. If the web service has more than one operation, then each operation may have its own input type, output type and fault type.

The data types can be declared in any language you like, as long as your web service API supports it. Data types are often specified using XML Schema though, since XML Schema is a natural fit for XML structures.

Here is an example types element that uses XML Schema to define an input type, an output type, and a fault type:

<?xml version="1.0" encoding="utf-8" ?>

<description

xmlns="http://www.w3.org/ns/wsdl"

targetNamespace= "http://jenkov.com/MyService"

xmlns:tns= "http://jenkov.com/MyService"

xmlns:stns= "http://jenkov.com/MyService/schema"

. . . >

...

<**types**>

<xs:schema

xmlns:xs= "http://www.w3.org/2001/XMLSchema"

targetNamespace= "http://jenkov.com/MyService/schema"

xmlns:tns= "http://jenkov.com/MyService/schema"

>

<xs:element name="latestTutorialRequest"

type="typeLatestTutorialRequest"/>

<xs:complexType name="typeLatestTutorialRequest">

<xs:sequence>

<xs:element name="date" type="xs:date"/>

</xs:sequence>

</xs:complexType>

<xs:element name="latestTutorialResponse" type="xs:string"/>

<xs:element name="invalidDateError" type="xs:string"/>

</xs:schema>

<**/types**>

. . .

</description>

The **latestTutorialRequest** element is the input type. As you can see, this element is of type typeLatestTutorialRequest, and it can contain a date, telling which date to get the latest tutorial after.

The **latestTutorialResponse** element is the output type. The content of this element is just a string. This will contain the URL for the latest tutorial published after the given date, or an empty string if no tutorial was published after the given date.

The **invalidDateError** element is the fault type. The content of this element is just a string. This string can contain a textual explanation of why the date given in the input request was faulty.

**XML Schema: Top Level Elements Only**

In XML Schema you can define all kinds of elements and types. However, only elements declared as single elements (there can be only one), and as top level elements (not nested inside other elements), can be referred to by the WSDL 2.0 interface and operationelements. In other words, you could not use the date element by itself as an input or output type of an operation, since it is declared inside the latestTutorialRequest.

**The Schema Name Space**

Notice how the example shown earlier declared a name space with the prefix stns. The Schema Target Name Space (stns). This name space URI is also referenced as the target name space of the XML Schema, but here it is declared via targetNameSpace and tns, not stns.

The prefixes are not important though. The prefixes used inside XML Schema only exist within the Schema declaration, not outside. Therefore these targetNamespace and tns namespaces do not conflict with those defined in the description element.

**WSDL 2.0 - interface**

* [Interface Name](http://tutorials.jenkov.com/wsdl/interface.html#interface-name)
* [Fault](http://tutorials.jenkov.com/wsdl/interface.html#fault)
* [Operation](http://tutorials.jenkov.com/wsdl/interface.html#operation)

|  |
| --- |
|  |

The WSDL interface element describes the operations supported by your web service. Each operation represents an interaction between the client and the service. In other words, the client can only call one operation per request. Operations are thus very similar to methods / procedures in a programming language.

Here is an interface element example:

<**interface** name = "latestTutorialInterface" >

<fault name = "invalidDateFault" element = "stns:invalidDateError"/>

<operation name="latestTutorialOperation"

pattern="http://www.w3.org/ns/wsdl/in-out"

style="http://www.w3.org/ns/wsdl/style/iri"

wsdlx:safe = "true">

<input messageLabel="In" element="stns:latestTutorialRequest" />

<output messageLabel="Out" element="stns:latestTutorialResponse" />

<outfault messageLabel="Out" ref ="tns:invalidDateFault" />

</operation>

</**interface**>

Note, this code style="http://www.w3.org/ns/wsdl/style/iri" has been reported to cause some validation problems with the above code. If you get them too, try to remove this line and try again.

I'll take you through the parts of the interface element in the rest of this text.

**Interface Name**

Since you can have more than one interface element in a WSDL 2.0 file, each interface must be given an unique name. This name is defined in the name attribute of the interface element, as shown here:

<interface **name="latestTutorialInterface"** >

**Fault**

The fault element defines a fault which can be sent back to the client. The same fault can be used by multiple operations. That is why it is defined outside the following operation element. Here is a fault element example:

<fault name = "invalidDateFault" element = "stns:invalidDateError"/>

The name attribute is used to give the fault a name. That name is used later in the operation element to reference this fault.

The element attribute contains a reference to a fault element, defined in the types element, earlier in the WSDL file. The stns name space (Schema Target Name Space) points inside the XML Schema defined in the types element. The invalidDateError is an element defined inside that XML Schema. Hence the qualified reference stns:invalidDateError .

**Operation**

The operation element describes an operation. A method or procedure, in other words. Here is an example:

<operation

name="latestTutorialOperation"

pattern="http://www.w3.org/ns/wsdl/in-out"

style="http://www.w3.org/ns/wsdl/style/iri"

wsdlx:safe = "true">

<input messageLabel="In" element="stns:latestTutorialRequest" />

<output messageLabel="Out" element="stns:latestTutorialResponse" />

<outfault messageLabel="Out" ref ="tns:invalidDateFault" />

</operation>

The name attribute of the operation element defines the name for the operation. This name must be unique within the interface element. The operation name is used later, when describing bindings for the operation.

The pattern attribute of the operation element describes what message exchange pattern this operation uses. These can be either in, out or in-out, meaning data in, data out, data in-out. Or, request-only, response-only, request-response. In-out is the most common pattern to use.

The style attribute is left out here.

The wsdlx:safe attribute indicates that this operation is safe to call, meaning the customer does not agree to buy anything, or order anything.

The input element describes the expected input data for the operation. The input element references an XML element defined in the typeselement of the WSDL file.

The output element describes the data returned by the operation. Theoutput element references an XML element defined in the types element of the WSDL file.

The outfault defines a possible output fault which can be sent to the client, if the operation fails. The ref attribute references the fault defined earlier in the interface element. The messageLabel attribute is signaling which message the fault replaces.

**WSDL 2.0 - binding**

|  |
| --- |
|  |

The WSDL binding element describes how your web service is bound to a protocol. In other words, how your web service is accessible. To be accessible, the web service must be reachable using some network protocol. This is called "binding" the web service to the protocol. And this is what the binding element does.

Here is a binding example:

<**binding** name="latestTutorialSOAPBinding"

interface="tns:latestTutorialInterface"

type="http://www.w3.org/ns/wsdl/soap"

wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">

<fault ref="tns:invalidDateFault" wsoap:code="soap:Sender"/>

<operation ref="tns:latestTutorialOperation"

wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>

</**binding**>

The binding name is referenced by the service element. Thus it should be unique within the WSDL file.

The interface attribute should refer to the name of an interface element defined in this WSDL file. Hence the tns: prefix (in this WSDL's Target Name Space).

The type attribute tells what kind of message format the interface is bound to. The value in the example states that the message format is SOAP.

The wsoap:protocol attribute specifies a SOAP binding - in other words, how the SOAP messages are transported. The value in the example specifies HTTP.

The operation element references an operation defined in the interface that this binding refers to. This references is made by the ref attribute. Notice the tns: prefix again, pointing to a name defined inside this WSDL.

The wsoap:mep attribute of the operation element, specifies a Message Exchange Pattern (MEP), which is a SOAP thing. See the the WSDL spec for more detail on this attribute.

The fault element defines a fault which may be sent back by the web service, via this binding. The fault element references a fault defined in the interface element this binding refers to. See the WSDL spec for more detail on this attribute.

**WSDL 2.0 - service**

|  |
| --- |
|  |

The WSDL service element describes the endpoint of your web service. In other words, the address where the web service can be reached.

Here is a service example:

<**service**

name ="latestTutorialService"

interface="tns:latestTutorialInterface">

<endpoint name ="latestTutorialEndpoint"

binding ="tns:latestTutorialSOAPBinding"

address ="http://jenkov.com/latestTutorial"/>

</**service**>

The name attribute describes the name of the service you are defining with the service element.

The interface attribute specifies which interface element this service element is related to.

The endpoint element describes the address of the web service. The endpoint binding attribute describes what binding element this endpoint uses. In other words, the protocol via which you can access the service. The address attribute describes the URI at which you can access the service.

Second Opinion of WSDL

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> ...

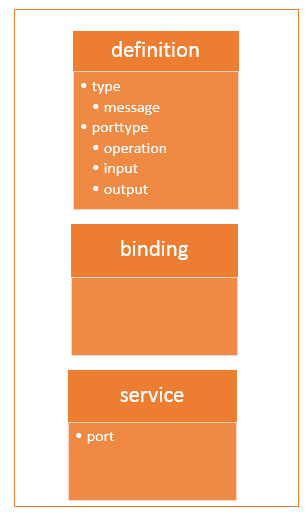
<!-- concrete definitions -->

<binding> ...

<service> ...

</definition>

Below is a diagram on the structure of a WSDL file

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

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 for 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.

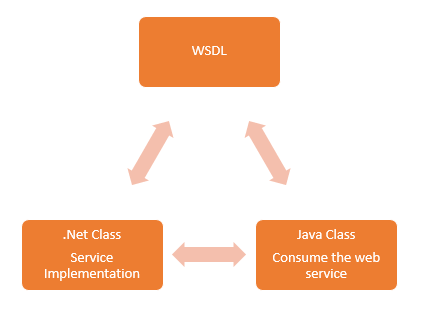
For example, there could be a data type called EmployeeDataType which could have 2 elements called "EmployeeName" of type string and "EmployeeID" of type number or integer. Together they form a data structure which then becomes a complex data type.

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**

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.

[](https://cdn.guru99.com/images/3-2016/032316_0742_WSDLWebserv2.png)

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.

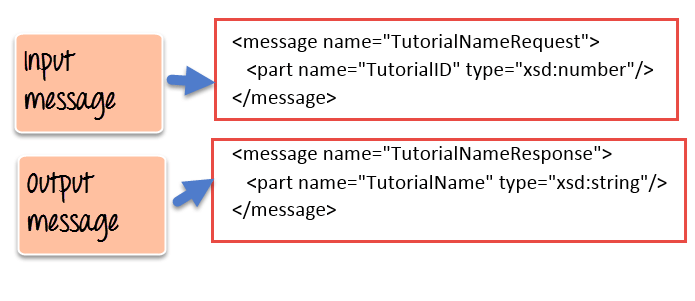
This element is basically used to describe the data that gets exchanged between the web service and the client application.

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://cdn.guru99.com/images/3-2016/032316_0742_WSDLWebserv3.png)

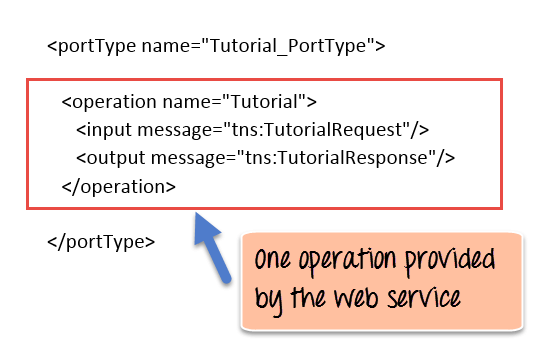
1. As we can see the web service has 2 messages, one for the input and the other for the output.
2. The input message is known as TutorialNameRequest which has one parameter called TutorialID. This parameter is of the type number which is specified by the xsd:number type
3. The output message is known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

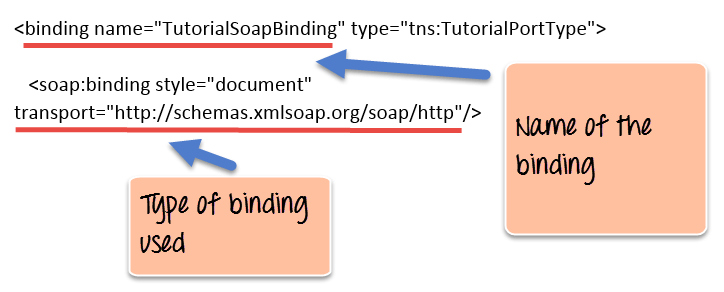
WSDL provides an element called **<portType>** which is used to define the operations provided by the Web service.

[](https://cdn.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://cdn.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.

**WSDL creation**

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