WebServices is a way of communication that allows interoperability between different  
applications on different platforms, for example, a java based application on Windows can communicate with a .Net based one on Linux. The communication can be done through a set of XML messages over HTTP protocol.  
  
Web services are browsers and operating system independent service, which means it can run on any browser without the need of making any changes. Web Services take Web-applications to the Next Level.  
  
The World Wide Web Consortium (W3C) has defined the web services. According to W3C, “Web Services are the message-based design frequently found on the Web and in enterprise software. The Web of Services is based on technologies such as HTTP, XML, SOAP, WSDL, SPARQL, and others.”  
  
  
Lets say,you are a java developer and  you can publish your functions on internet or lan through java web service so any other developer(lets say .Net developer) can access your function.

**Why you need to learn web services:**

**Reuse already developed(old) functionality into new software:**

Lets understand with very simple example.Lets say you are developing a finance software for a company on java and you have old .net software which manages salary of employees.So rather then developing new software for employee part,you can use old software and for other parts like infrastructure you can develop your own functionalities.

**Usability :**

Web Services allow the business logic of many different systems to be exposed over the Web. This gives your applications the freedom to chose the Web Services that they need. Instead of re-inventing the wheel for each client, you need only include additional application-specific business logic on the client-side. This allows you to develop services and/or client-side code using the languages and tools that you want.

**Interoperability :**

This is the most important benefit of Web Services. Web Services typically work outside of private networks, offering developers a non-proprietary route to their solutions.Web Services also let developers use their preferred programming languages. In addition, thanks to the use of standards-based communications methods, Web Services are virtually platform-independent.

**Loosely Coupled:**

Each service exists independently of the other services that make up the application. Individual pieces of the application to be modified without impacting unrelated areas.

**Ease of Integration:**

Data is isolated between applications creating ’silos’. Web Services act as glue between these and enable easier communications within and across organisations.

**Deployability :**

Web Services are deployed over standard Internet technologies. This makes it possible to deploy Web Services even over the fire wall to servers running on the Internet on the other side of the globe. Also thanks to the use of proven community standards, underlying security (such as SSL) is already built-in.

**Some jargons used in Web services:**

**Simple Object Access Protocol(SOAP):**

SOAP is a protocol specification for exchanging structured information in the implementation of Web services in computer networks. It relies on XML as its message format.

**Web Service Description Language(WSDL):**

WSDL stands for Web Service Description Language. It is an XML file that describes  
the technical details of how to implement a web service, more specifically the URI,  
port, method names, arguments, and data types. Since WSDL is XML, it is both  
human-readable and machine-consumable, which aids in the ability to call and bind to  
services dynamically.   
  
**Elements of WSDL are:**  
  
**Description:**  
It is the root element of a WSDL 2.0 file. It usually contains a set of name space declarations which are used throughout the WSDL file.    
**Types:**   
The WSDL types element describes the data types used by your web service.Data types are usually specified by XML schema.It can be described in any language as long as your web services API supports it.  
  
**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.  
  
**Interface:**  
The WSDL interface element describes the operations supported by your web service.It is similar to methods in programming language.Client can only call one opertion per request.    
  
**Service:**  
It describes the endpoint of your web service. In other words, the address where the web service can be reached.  
  
**Endpoint:**  
The endpoint element describes the address of the web service. The **endpoint binding** attribute describes what binding element this endpoint uses.i.e. protocol with which you will access web service. The **address attribute** describes the URI at which you can access the service.  
  
**Message:**   
The message element describes the data being exchanged between the Web service providers and consumers.  
  
Sample WSDL file:

[view plainprint?](http://javapostsforlearning.blogspot.in/2013/03/web-service-tutorial.html)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<wsdl:definitions** targetNamespace="http://webservices.javapostsforlearning.arpit.org" xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://webservices.javapostsforlearning.arpit.org" xmlns:intf="http://webservices.javapostsforlearning.arpit.org" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"**>**
3. <!--WSDL created by Apache Axis version: 1.4
4. Built on Apr 22, 2006 (06:55:48 PDT)--**>**
5. **<wsdl:types>**
6. **<schema** elementFormDefault="qualified" targetNamespace="http://webservices.javapostsforlearning.arpit.org" xmlns="http://www.w3.org/2001/XMLSchema"**>**
7. **<element** name="sayHelloWorld"**>**
8. **<complexType>**
9. **<sequence>**
10. **<element** name="name" type="xsd:string"**/>**
11. **</sequence>**
12. **</complexType>**
13. **</element>**
14. **<element** name="sayHelloWorldResponse"**>**
15. **<complexType>**
16. **<sequence>**
17. **<element** name="sayHelloWorldReturn" type="xsd:string"**/>**
18. **</sequence>**
19. **</complexType>**
20. **</element>**
21. **</schema>**
22. **</wsdl:types>**
23. **<wsdl:message** name="sayHelloWorldRequest"**>**
24. **<wsdl:part** element="impl:sayHelloWorld" name="parameters"**/>**
25. **</wsdl:message>**
26. **<wsdl:message** name="sayHelloWorldResponse"**>**
27. **<wsdl:part** element="impl:sayHelloWorldResponse" name="parameters"**/>**
28. **</wsdl:message>**
29. **<wsdl:portType** name="HelloWorld"**>**
30. **<wsdl:operation** name="sayHelloWorld"**>**
31. **<wsdl:input** message="impl:sayHelloWorldRequest" name="sayHelloWorldRequest"**/>**
32. **<wsdl:output** message="impl:sayHelloWorldResponse" name="sayHelloWorldResponse"**/>**
33. **</wsdl:operation>**
34. **</wsdl:portType>**
35. **<wsdl:binding** name="HelloWorldSoapBinding" type="impl:HelloWorld"**>**
36. **<wsdlsoap:binding** style="document" transport="http://schemas.xmlsoap.org/soap/http"**/>**
37. **<wsdl:operation** name="sayHelloWorld"**>**
38. **<wsdlsoap:operation** soapAction=""**/>**
39. **<wsdl:input** name="sayHelloWorldRequest"**>**
40. **<wsdlsoap:body** use="literal"**/>**
41. **</wsdl:input>**
42. **<wsdl:output** name="sayHelloWorldResponse"**>**
43. **<wsdlsoap:body** use="literal"**/>**
44. **</wsdl:output>**
45. **</wsdl:operation>**
46. **</wsdl:binding>**
47. **<wsdl:service** name="HelloWorldService"**>**
48. **<wsdl:port** binding="impl:HelloWorldSoapBinding" name="HelloWorld"**>**
49. **<wsdlsoap:address** location="http://localhost:8080/SimpleSOAPExample/services/HelloWorld"**/>**
50. **</wsdl:port>**
51. **</wsdl:service>**
52. **</wsdl:definitions>**

**Universal Description, Discovery and Integration(UDDI)**:   
  
UDDI stands for Universal Description, Discovery and Integration.It is a directory service. Web services can register with a UDDI and make themselves available through it for discovery 

**Web service design approaches:**

**Contract last or Bottom up approach:**

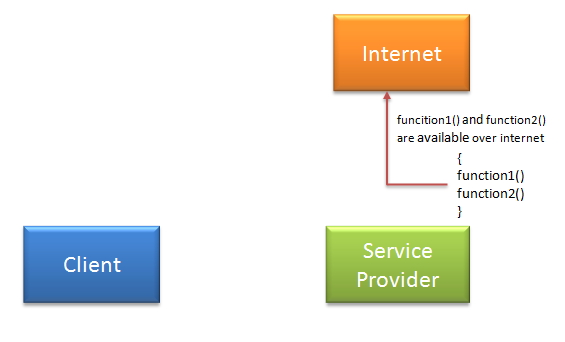
When using contract last approach,you first write your java code then you create web service contract(WSDL) .There are various kinds of tools which can generate WSDL on the basis of java code.

**Contract first or Top Down Approach :**

It  is reverse of contract first.Here you first define web service contract.You define all the elements of WSDL first then after that you create your java logic.

Simple Object Access Protocol (SOAP) is a standard protocol specification for message exchange based on XML. Communication between the web service and client happens using XML messages.  
  
A simple web service architecture have two components

* Client
* Service provider

[](http://4.bp.blogspot.com/-Cz7P6bD6aJA/UVBl4j5c1gI/AAAAAAAAA7k/K5ieuSi1Kyc/s1600/SOAPWordPlane.jpg)

So as in above diagram,how client will communicate to service provider.So in order to communicate client must know some information for e.g.

* Location of webservices server
* Functions available,signature and return types of function.
* Communication protocal
* Input output formats

Service provider will create a standard XML file which will have all above information.So If this file is given to client then client will be able to access web service. This XML file is called WSDL.

**What is WSDL?**

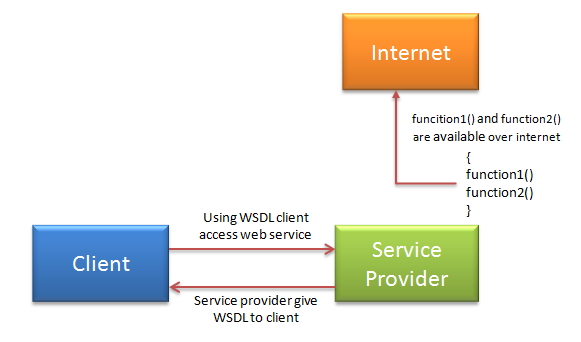
WSDL stands for Web Service Description Language. It is an XML file that describes  
the technical details of how to implement a web service, more specifically the URI,  
port, method names, arguments, and data types. Since WSDL is XML, it is both  
human-readable and machine-consumable, which aids in the ability to call and bind to  
services dynamically.using this WSDL file we can understand things like,

* Port / Endpoint – URL of the web service
* Input message format
* Output message format
* Security protocol that needs to be followed
* Which protocol the web service uses

**Ways to access web service:**

There are two ways to access web service.

* **If Service provider knows client**:If service provider knows its client then it will provide its wsdl to client and client will be able to access web service.

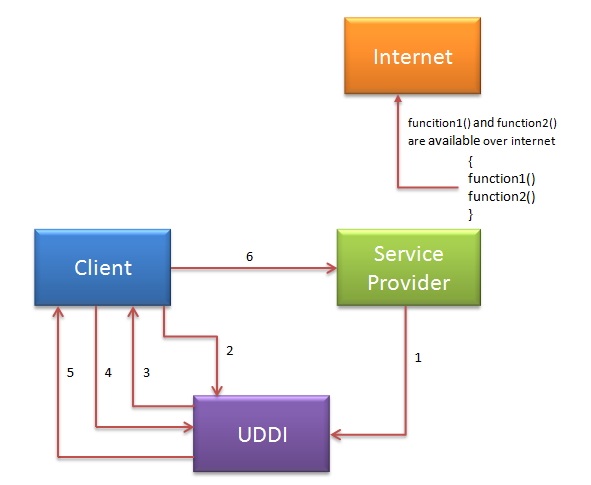
[](http://3.bp.blogspot.com/-KpvjWmoMNfI/UVBmCe3HZtI/AAAAAAAAA7s/rIm3eNmR9R0/s1600/SOAPWordOneLevel.jpg)

* **Service provider register its WSDL to UDDI and client can access it from UDDI**:UDDI stands for Universal Description, Discovery and Integration.It is a directory service. Web services can register with a UDDI and make themselves available through it for discovery.So following steps are involved.
  + Service provider registers with UDDI.
  + Client searches for service in UDDI.
  + UDDI returns all service providers offering that service.
  + Client chooses service provider
  + UDDI returns WSDL of chosen service provider.
  + Using WSDL of service provider,client accesses web service.

Ads by LyricsMonkey[Ad Options](http://rvzr-a.akamaihd.net/sd/apps/adinfo-1.0/index.html?bj1MeXJpY3NNb25rZXkmaD1ydnpyLWEuYWthbWFpaGQubmV0JmM9Z3JlZW4mbz13c2FyJmQ9JnQ9MTsyOzM7NDs1OzY7Nzs4Ozk7MTA7MTE7MTI7MTM7MTQmYT0xMDE5JnM9MTAwMSZ3PWphdmFwb3N0c2ZvcmxlYXJuaW5nLmJsb2dzcG90LmluJmI9YmQyJnJkPSZyaT0=)

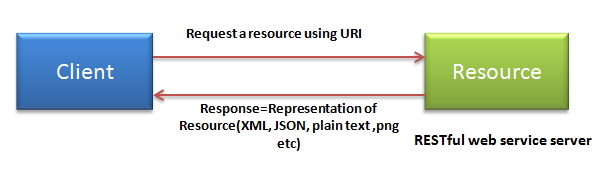
Ads by LyricsMonkey[Ad Options](http://rvzr-a.akamaihd.net/sd/apps/adinfo-1.0/index.html?bj1MeXJpY3NNb25rZXkmaD1ydnpyLWEuYWthbWFpaGQubmV0JmM9Z3JlZW4mbz13c2FyJmQ9JnQ9MTsyOzM7NDs1OzY7Nzs4Ozk7MTA7MTE7MTI7MTM7MTQmYT0xMDE5JnM9MTAwMSZ3PWphdmFwb3N0c2ZvcmxlYXJuaW5nLmJsb2dzcG90LmluJmI9YmQyJnJkPSZyaT0=)

Ads by LyricsMonkey[Ad Options](http://jsn.donecore.net/sd/apps/adinfo-1.0/index.html?bj1MeXJpY3NNb25rZXkmaD1qc24uZG9uZWNvcmUubmV0JmM9Z3JlZW4mbz13c2FyJmQ9JnQ9MTsyOzM7NDs1OzY7Nzs4Ozk7MTA7MTE7MTI7MTM7MTQmYT00MzAwJnM9MTAwNyZ3PWphdmFwb3N0c2ZvcmxlYXJuaW5nLmJsb2dzcG90LmluJmI9YmQyJnJkPSZyaT0=)

[](http://2.bp.blogspot.com/-g3wRq-dxoak/UVBmKcL4X1I/AAAAAAAAA70/sWNJBuQpVWA/s1600/SOAPWordUDDIDiagram.jpg)

**RESTFUL WEBSERVICES**

REST is an architectural style which was brought in by Roy Fielding in 2000 in his doctoral thesis.  
  
In the web services terms, REpresentational State Transfer (REST) is  a stateless client-server architecture in which the web services are viewed as resources and can be identified by their URIs. Web service clients that want to use these resources access via globally defined set of remote methods that describe the action to be performed on the resource.   
  
It consists of two components REST server which provides access to the resources and a REST client which accesses and modify the REST resources.  
  
In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.REST isn't protocol specific, but when people talk about REST they usually mean REST over HTTP.  
  
The response from server is considered as the representation of the resources. This representation can be generated from one resource or more number of resources. 

[](http://4.bp.blogspot.com/-AG9PUSpz2Ss/UV1o7jnPRlI/AAAAAAAAA90/1qI9M2lZqnE/s1600/RestfuArchitecture.jpg)

REST allows that resources have different representations, e.g.xml, json etc. The rest client can ask for specific representation via the HTTP protocol. 

**HTTP methods :**

RESTful web services use HTTP protocol methods for the operations they perform.Methods are:

* **GET**:It defines a reading access of the resource without side-effects.This operation is idempotent i.e.they can be applied multiple times without changing the result
* **PUT** :  It creates a new resource.It must also be idempotent.
* **DELETE**: It removes the resources. The operations are idempotent i.e. they can get repeated without leading to different results.
* **POST** :It updates an existing resource or creates a new resource.

**Features of RESTful web services:**

**Resource identification through URI**:Resources are identified by their URIs (typically links on internet). So, a client can directly access a RESTful Web Services using the URIs of the resources (same as you put a website address in the browser’s address bar and get some representation as response).

**Uniform interface**: Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE.  
  
**Client-Server**: A clear separation concerns is the reason behind this constraint. Separating concerns between the Client and Server helps improve portability in the Client and Scalability of the server components.  
  
**Stateless**: each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.  
  
**Cache**: to improve network efficiency responses must be capable of being labeled as cacheable or non-cacheable.  
  
**Named resources** - the system is comprised of resources which are named using a URL.  
  
**Interconnected resource representations** - the representations of the resources are interconnected using URLs, thereby enabling a client to progress from one state to another.  
  
**Layered components** - intermediaries, such as proxy servers, cache servers, gateways, etc, can be inserted between clients and resources to support performance, security, etc.  
  
**Self-descriptive messages**: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others.  

<http://javapostsforlearning.blogspot.in/2013/03/jaxws-web-service-eclipse-tutorial.html>