1. [**Web services(flow)**](#Webservice_diagram_flow)
2. [**diff\_httppost\_put**](#diff_httppost_put)
3. [**JSON\_VS\_XML**](#JSON_VS_XML)
4. [**Wsdl file**](#wsdl_file)
5. [**SOAP\_WSDl\_BINDING\_STYLE**](#SOAP_WSDl_BINDING_STYLE)
6. [**diff\_SOAPws\_RESTfulws**](#diff_SOAPws_RESTfulws) **(**REST supports different format like **text, JSON and XML** while **SOAP only support XML**.
7. [**decide\_webservice\_to\_use**](#decide_webservice_to_use)**?**
8. [**Approaches\_Develop\_SOAP\_bASED**](#Approaches_Develop_SOAP_bASED)
9. [**tools\_to\_testWS**](#tools_to_testWS)
10. [**diff\_SOA\_Webservice**](#diff_SOA_Webservice)
11. [**REST WEBSERVICE**](#REST_WEBSERVICE)
12. [**http\_methods\_Supporteby\_RESTFULL**](#http_methods_Supporteby_RESTFULL)**(GET,POST,PUT , DELETE)**
13. [**diff\_SpringMVC\_RESTWebservice**](#diff_SpringMVC_RESTWebservice)
14. [**types\_of\_operations\_available\_WSDL**](#types_of_operations_available_WSDL)
15. [**diff\_JAX\_WS\_axis2\_cxf**](#diff_JAX_WS_axis2_cxf)
16. [Building a RESTFul Service with Spring MVC](#build_rest_withmvc)
17. [**Call Rest Full Service From Java**](#CallRestWebServiceFromjava)
18. [**RESTful java web service with XML Response**](#restfull_return_xml)
19. [**Difference between Servlet and RestFul Services.**](#diff_Servlet_REST)
20. [**Test REST using JerseyTest**](#test_REST_jersey)
21. [**Secure web services**](#secure_REestFul)
22. [**How to implement a HTTPS login page in a web application?**](#implement_https)
23. [**REST OR SOAP WHICH ONE TO USE WHEN**](#rest_soap_whichonetouse)

So without further ado, some REST services:

* The Google Glass API, known as "Mirror API", is a pure REST API. Here is [an excellent video talk](http://www.youtube.com/watch?feature=player_embedded&v=JpWmGX55a40) about this API. (The actual API discussion starts after 16 minutes or so.)
* Twitter has a [**REST API**](https://dev.twitter.com/docs/api) (in fact, this was their original API and, so far as I can tell, it's still the main API used by Twitter application developers),
* [**Flickr**](http://www.flickr.com/services/api/),
* **Amazon.com** offer several REST services, e.g., for their [S3 storage solution](http://docs.aws.amazon.com/AmazonS3/2006-03-01/API/APIRest.html),
* [**Atom**](http://en.wikipedia.org/wiki/Atom_(standard)) is a RESTful alternative to RSS,
* [**Tesla Model S**](http://docs.timdorr.apiary.io/) uses an (undocumented) REST API between the car systems and its Android/iOS apps.

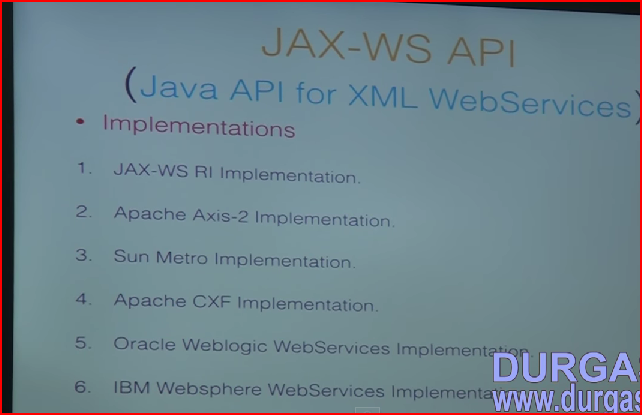
**Types of Webservices**

1. **JAX-RPC AI**

**Implementations:Apache Axis**

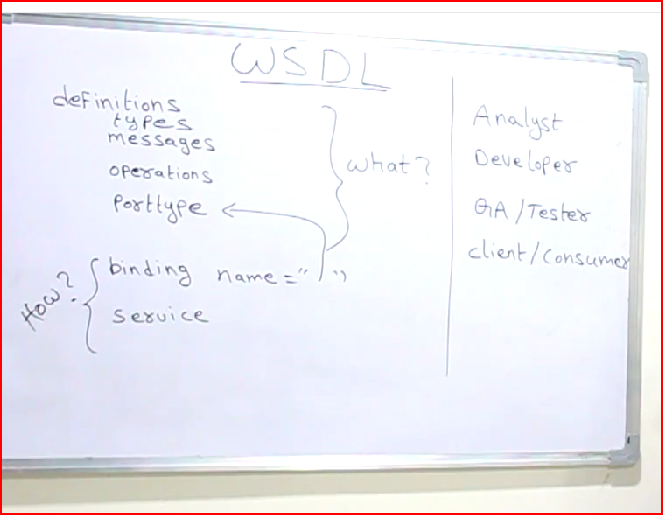
1. **JAX-WS**

**Implementations: Apache Axis2**

****

**SOA Architecture:**

A simple definition: A web service is a function that can be accessed by other programs over the web (Http). To clarify a bit, when you create a website in PHP that outputs HTML its target is the browser and by extension the human being reading the page in the browser. A web service is not targeted at humans but rather at other programs.



<http://www.tutorialspoint.com/wsdl/wsdl_example.htm>

## Content of HelloService.wsdl file

|  |
| --- |
| <definitions name="HelloService"  targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"  xmlns="http://schemas.xmlsoap.org/wsdl/"  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"  xmlns:xsd="http://www.w3.org/2001/XMLSchema">    <message name="SayHelloRequest">  <part name="firstName" type="xsd:string"/>  </message>  <message name="SayHelloResponse">  <part name="greeting" type="xsd:string"/>  </message>  <portType name="Hello\_PortType">  <operation name="sayHello">  <input message="tns:SayHelloRequest"/>  <output message="tns:SayHelloResponse"/>  </operation>  </portType>  <binding name="Hello\_Binding" type="tns:Hello\_PortType">  <soap:binding style="rpc"  transport="http://schemas.xmlsoap.org/soap/http"/>  <operation name="sayHello">  <soap:operation soapAction="sayHello"/>  <input>  <soap:body  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"  namespace="urn:examples:helloservice"  use="encoded"/>  </input>  <output>  <soap:body  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"  namespace="urn:examples:helloservice"  use="encoded"/>  </output>  </operation>  </binding>  <service name="Hello\_Service">  <documentation>WSDL File for HelloService</documentation>  <port binding="tns:Hello\_Binding" name="Hello\_Port">  <soap:address  location="http://www.examples.com/SayHello/">  </port>  </service>  </definitions> |

**WSDL Example Tutorial Test Sample SOAP Web Service WSDL**

# July 21, 2011

This sample web service would hopefully give you all a very good idea on how to do it yourself.

## What is WSDL

**The Web Services Description Language or WSDL** for short describes a the Web service interface. It consists of messages that are exchanged between the client and server. The messages are described abstractly and then bound to a concrete network protocol and message format. Web service definitions can be mapped to any implementation language, platform, object model, or messaging system.

## [WSDL Example Tutorial Test Sample SOAP Web Service WSDL](http://www.teqlog.com/wp-content/uploads/2011/07/wsdl_structure.gif)WSDL Elements

### *****Definition*****

This contains the attribute name, which in turn has the web service name.

### *****Types*****

This element uses the XML schema language to declare complex data types and elements that are used elsewhere in the WSDL document.

### *****Message*****

This element describes the message’s payload using XML schema built-in types, complex types.

### *****PortType/Interface and Operation*****

These elements describe a Web service’s interface and define its methods.

### *****Binding*****

This element assigns a portType and its operation elements to a particular protocol (for instance, SOAP 1.1) and encoding style.

### *****Service*****

This element is responsible for assigning an Internet address to a specific binding.

## ****WSDL Example 1****

Web Service Name : WebServiceOne

This WSDL example contains:

- SOAP Protocol

- Document Literal Binding

- No optional SOAP Header included in WSDL

- SOAP Body

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

<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

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

xmlns:tns="http://www.teknocrat.com/WebServiceOne/"

xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" name="WebServiceOne"

targetnamespace="http://www.teknocrat.com/WebServiceOne/">

<wsdl:types>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetnamespace="http://www.teknocrat.com/WebServiceOne/">

<xsd:element name="WebServiceOneOperation">

<xsd:complextype>

<xsd:sequence>

<xsd:element name="in" type="xsd:string"></xsd:element>

</xsd:sequence>

</xsd:complextype>

</xsd:element>

<xsd:element name="WebServiceOneOperationResponse">

<xsd:complextype>

<xsd:sequence>

<xsd:element name="out" type="xsd:string"></xsd:element>

</xsd:sequence>

</xsd:complextype>

</xsd:element></xsd:schema></wsdl:types>

<wsdl:message name="WebServiceOneOperationRequest">

<wsdl:part name="parameters" element="tns:WebServiceOneOperation"></wsdl:part>

</wsdl:message>

<wsdl:message name="WebServiceOneOperationResponse">

<wsdl:part name="parameters" element="tns:WebServiceOneOperationResponse"></wsdl:part>

</wsdl:message>

<wsdl:porttype name="WebServiceOnePortType">

<wsdl:operation name="WebServiceOneOperation">

<wsdl:input message="tns:WebServiceOneOperationRequest"></wsdl:input>

<wsdl:output message="tns:WebServiceOneOperationResponse"></wsdl:output>

</wsdl:operation>

</wsdl:porttype>

<wsdl:binding name="WebServiceOneBinding" type="tns:WebServiceOnePortType">

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

<wsdl:operation name="WebServiceOneOperation">

<soap:operation soapaction="http://www.teknocrat.com/WebServiceOne/WebServiceOneOperation" />

<wsdl:input>

<soap:body use="literal" />

</wsdl:input>

<wsdl:output>

<soap:body use="literal" />

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

<wsdl:service name="WebServiceOne">

<wsdl:port name="WebServiceOnePort" binding="tns:WebServiceOneBinding">

<soap:address location="http://www.teknocrat.com/" />

</wsdl:port>

</wsdl:service></wsdl:definitions>

# Analysis of the Example

* **Definition :**HelloService
* **Type :** Using built-in data types and they are defined in XMLSchema.
* **Message :**
  1. sayHelloRequest : firstName parameter
  2. sayHelloresponse: greeting return value
* **Port Type:** sayHello **operation** that consists of a request and response service.
* **Binding:** Direction to use the SOAP HTTP transport protocol.
* **Service:** Service available at http://www.examples.com/SayHello/.
* **Port:** Associates the binding with the URI http://www.examples.com/SayHello/ where the running service can be accessed.

### WSDL & SOAP: Binding Style to use - Document or RPC?

#### By gopalan on [Jul 08, 2006](https://blogs.oracle.com/gopalan/entry/wsdl_soap_binding_style_to)

There are two communication *style*models that are used to translate a WSDL binding to a SOAP message body. They are:

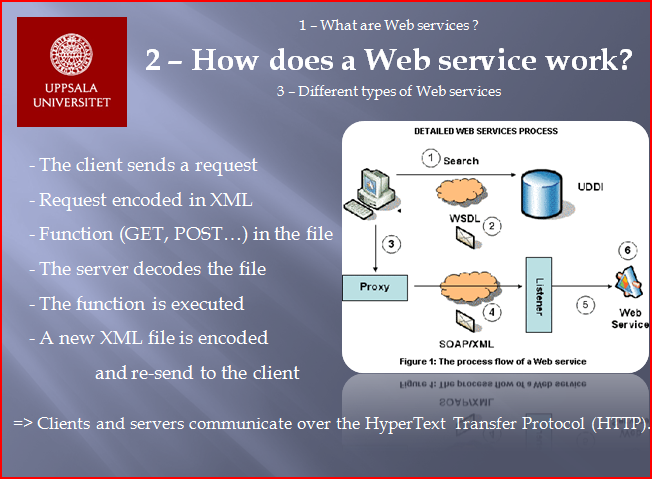
* *Document, and*
* *RPC*

The advantage of using a *Document*style model is that you can structure the SOAP body any way you want it as long as the content of the SOAP message body is any arbitrary XML instance. The *Document*style is also referred to as*Message-Oriented style*.   
  
However, with an *RPC*style model, the structure of the SOAP request body must contain both the operation name and the set of method parameters. The *RPC*style model assumes a specific structure to the XML instance contained in the message body.  
  
Furthermore, there are two encoding *use*models that are used to translate a WSDL binding to a SOAP message. They are:

* *literal, and*
* *encoded*

For RPC style web services, the type information includes the standard [XML data types](http://www.javaexperience.com/strip-invalid-characters-from-xml/) like String, int, long etc. But for DOCUMENT style web services, the types element refers to an XSD which can be used to describe complex data types by using XSD constructs. You may want to read about [RPC vs DOCUMENT web services](http://www.javaexperience.com/jax-ws-rpc-vs-document-style-web-services/). In our WSDL, StockPriceImplService\_schema1.xsd is the XSD which contains the data type information for each message. The path of XSD could be absolute or relative to the WSDL.  
  
Read more: <http://www.javaexperience.com/understanding-wsdl-structure-and-elements/#ixzz3Przardrm>

Web services:



What is SOAP

**SOAP is an XML-based protocol for exchanging information between computers.**

SOAP (Simple Object Access Protocol) is a way for a program running in one kind of [operating system](http://searchcio-midmarket.techtarget.com/definition/operating-system) (such as [Windows 2000](http://searchenterprisedesktop.techtarget.com/definition/Windows-2000)) to communicate with a program in the same or another kind of an operating system (such as [Linux](http://searchenterpriselinux.techtarget.com/definition/Linux)) by using the World Wide Web's Hypertext Transfer Protocol ([HTTP](http://searchwindevelopment.techtarget.com/definition/HTTP))and its Extensible Markup Language ([XML](http://searchsoa.techtarget.com/definition/XML)) as the mechanisms for information exchange. Since Web [protocol](http://searchnetworking.techtarget.com/definition/protocol)s are installed and available for use by all major operating system platforms, HTTP and XML provide an already at-hand solution to the problem of how programs running under different operating systems in a network can communicate with each other. SOAP specifies exactly how to encode an HTTP header and an XML file so that a program in one computer can call a program in another computer and pass it information. It also specifies how the called program can return a response.

**Other frameworks, including CORBA, DCOM, and Java RMI, provide similar functionality to SOAP, but SOAP messages are written entirely in XML and are therefore uniquely platform- and language-independent.**

<http://javapostsforlearning.blogspot.com/2013/03/web-service-tutorial.html>

[JAX-WS standard](http://docs.oracle.com/javaee/5/tutorial/doc/bnayl.html) = JAVA API for XML Web Services

REST = Representational state transfer.

RESTful web services are based on the way how our web works. Our very own world wide web (www) – the largest distributed application – is based on an architectural style called REST – **Representational State Transfer**. REST is neither a standard nor a protocol. It is just an architectural style like say for example client-server architecture (client-server is neither a standard nor a protocol). Web services following this architectural style are said to be RESTful Web services.

<http://stackoverflow.com/questions/10557514/different-types-of-webservices-in-java-where-to-start>

types of web services:

1. [SOAP Web Services](http://en.wikipedia.org/wiki/SOAP) are standard-based and supported by almost every software platform: They rely heavily in XML and have support for transactions, security, asynchronous messages and many other issues. It’s a pretty big and complicated standard, but covers almost every messaging situation. On the other side, [RESTful services](http://en.wikipedia.org/wiki/Representational_state_transfer) relies of HTTP protocol and verbs (GET, POST, PUT, DELETE) to interchange messages in any format, preferable JSON and XML. It’s a pretty simple and elegant architectural approach.
2. As in every topic in the Java World, there are several libraries to build/consume Web Services. In the SOAP Side you have the [JAX-WS standard](http://docs.oracle.com/javaee/5/tutorial/doc/bnayl.html) and [Apache Axis](http://axis.apache.org/axis/), and in REST you can use [Restlets](http://www.restlet.org/) or [Spring REST Facilities](http://static.springsource.org/spring/docs/3.0.0.M3/spring-framework-reference/html/ch18s02.html) among other libraries.
3. [this article](http://www.infoq.com/articles/rest-soap-when-to-use-each) states that RESTful Services are appropriate in this scenarios:

* If you have limited bandwidth
* **If your operations are stateless**: No information is preserved from one invocation to the next one, and each request is treated independently.
* If your clients require caching

**While SOAP is the way to go when:**

* If you require asynchronous processing
* If you need formal contract/Interfaces
* **In your service operations are stateful:** For example, you store information/data on a request and use that stored data on the next one.

The SOAP WS is transport protocol neutral. Supports multiple protocols like HTTP(S), Messaging, TCP, UDP SMTP, etc. The REST is transport protocol specific. Supports only HTTP or HTTPS protocols.

**What are the differences between both SOAP WS and RESTful WS?    
A.**

**) What is differences between RESTful web services and SOAP web services ?**

Though both RESTful web series and SOAP web service can operate cross platform they are architecturally different to each other, here is some of differences between REST and SOAP:

1) REST is more simple and easy to use than SOAP

2) REST uses HTTP protocol for producing or consuming web services while SOAP uses XML.

3) REST is lightweight as compared to SOAP and preferred choice in mobile devices and PDA's.

4) REST supports different format like **text, JSON and XML** while **SOAP only support XML**.

5) REST web services call can be cached to improve performance.

* The SOAP WS supports both remote procedure call (i.e. RPC) and message oriented middle-ware (MOM) integration styles. The Restful Web Service supports only RPC integration style.
* The SOAP WS is transport protocol neutral. Supports multiple protocols like HTTP(S),  Messaging, TCP, UDP SMTP, etc. The REST is transport protocol specific. Supports only HTTP or HTTPS protocols.

* The SOAP WS permits only XML data format. You define operations, which tunnels through the POST. The focus is on accessing the named operations and exposing the application logic as a service. The REST permits multiple data formats like XML, JSON data, text, HTML, etc. Any browser can be used because the REST approach uses the standard GET, PUT, POST, and DELETE Web operations. The focus is on accessing the named resources and exposing the data as a service. REST has AJAX support. It can use the XMLHttpRequest object. Good for stateless CRUD (Create, Read, Update, and Delete) operations.   
    
           GET - read()  
           POST - create()  
           PUT - update()  
           DELETE - delete()
* SOAP based reads cannot be cached. REST based reads can be cached. Performs and scales better.
* SOAP WS supports both SSL security and WS-security, which adds some enterprise security features like maintaining security right up to the point where it is needed, maintaining identities through intermediaries and not just point to point SSL only, securing different parts of the message with different security algorithms, etc. The REST supports only point-to-point SSL security. The SSL encrypts the whole message, whether all of it is sensitive or not.
* The SOAP has comprehensive support for both ACID based  transaction management  for short-lived transactions and compensation based transaction management for long-running transactions. It also supports two-phase commit across distributed resources. The REST supports transactions, but it  is neither ACID compliant nor can provide two phase commit across distributed transactional resources as it is limited by its HTTP protocol.
* The SOAP has success or retry logic built in and provides end-to-end reliability even through SOAP intermediaries. REST does not have a standard messaging system, and expects clients invoking the service to deal with communication failures by retrying.

<http://www.infoq.com/articles/rest-soap-when-to-use-each>

So this means areas that REST works really well for are:

* **Limited bandwidth and resources;** remember the return structure is really in any format (developer defined). Plus, any browser can be used because the REST approach uses the standard *GET*, *PUT*, *POST*, and *DELETE* verbs. Again, remember that REST can also use the *XMLHttpRequest* object that most modern browsers support today, which adds an extra bonus of AJAX.
* **Totally stateless operations;** if an operation needs to be continued, then REST is not the best approach and SOAP may fit it better. However, if you need stateless CRUD (Create, Read, Update, and Delete) operations, then REST is it.
* **Caching situations;** if the information can be cached because of the totally stateless operation of the REST approach, this is perfect.

That covers a lot of solutions in the above three. So why would I even consider SOAP? Again, SOAP is fairly mature and well-defined and does come with a complete specification. The REST approach is just that, an approach and is wide open for development, so if you have the following then SOAP is a great solution:

* **Asynchronous processing and invocation;** if your application needs a guaranteed level of reliability and security then SOAP 1.2 offers additional standards to ensure this type of operation. Things like WSRM – WS-Reliable Messaging.
* **Formal contracts;** if both sides (provider and consumer) have to agree on the exchange format then SOAP 1.2 gives the rigid specifications for this type of interaction.
* **Stateful operations**; if the application needs contextual information and conversational state management then SOAP 1.2 has the additional specification in the WS\* structure to support those things (Security, Transactions, Coordination, etc). Comparatively, the REST approach would make the developers build this custom plumbing.

http://java-success.blogspot.in/2012/02/java-web-services-interview-questions.html

**Q.** How would you decide what style of Web Service to use? SOAP WS or REST?  
**A**. In general, a REST based Web service is preferred due to its simplicity, performance, scalability, and support for multiple data formats. SOAP is favored where service requires comprehensive support for security and transactional reliability.  
  
The answer really depends on the functional and non-functional requirements. Asking the questions listed below will help you choose.

* **Does the service expose data or business logic?**(REST is a better choice for exposing data, SOAP WS might be a better choice for logic).Do the consumers and the service providers require a formal contract? (SOAP has a formal contract via WSDL)
* **Do we need to support multiple data formats?**
* **Do we need to make AJAX calls?**(REST can use the XMLHttpRequest)
* **Is the call synchronous or  asynchronous?**
* **Is the call stateful or stateless?** (REST is suited for statless CRUD operations)
* **What level of security is required?** (SOAP WS has better support for security)
* **What level of transaction support is required?**(SOAP WS has better support for transaction management)
* **Do we have limited band width?** (SOAP is more verbose)
* **What’s best for the developers who will build clients for the service?**(REST is easier to implement, test, and maintain)

**Q****.**What tools do you use to test your Web Services?  
**A.** **SoapUI** tool for SOAP WS and the Firefox "**poster**" plugin for RESTFul services.

**Q****.** What is the difference between SOA and a Web service?   
**A.**  
  
**SOA is**a software design principle and an architectural pattern for implementing loosely coupled, reusable and coarse grained services. You can implement SOA using any protocols such as HTTP, HTTPS, JMS, SMTP, RMI, IIOP (i.e. EJB uses IIOP), RPC etc. Messages can be in XML or Data Transfer Objects (DTOs).       
  
**Web service is** an implementation technology and one of the ways to implement SOA. You can build SOA based applications without using Web services – for example by using other traditional technologies like Java RMI, EJB, JMS based messaging, etc. But what Web services offer is the standards based  and platform-independent service via HTTP, XML, SOAP, WSDL and UDDI, thus allowing interoperability between heterogeneous technologies such as J2EE and .NET.

 Q. What are the different approaches to developing a SOAP based Web service? A. 2 approaches.  
  
The **contract-first** approach, where you define the contract first with XSD and WSDL and the generate the Java classes from the contract.

* The **contract-last**approach where you  define the Java classes first and then generate the contract, which is the  WSDL file from the Java classes.

<http://javarevisited.blogspot.com.au/2012/01/rest-web-services-framework-interview.html>

**2) What is differences between RESTful web services and SOAP web services ?**

Though both RESTful web series and SOAP web service can operate cross platform they are architecturally different to each other, here is some of differences between REST and SOAP:

1) REST is more simple and easy to use than SOAP

2) REST uses HTTP protocol for producing or consuming web services while SOAP uses XML.

3) REST is lightweight as compared to SOAP and preferred choice in mobile devices and PDA's.

4) REST supports different format like **text, JSON and XML** while **SOAP only support XML**.

5) REST web services call can be cached to improve performance.

**3) What is Restlet framework ?**

Restlet is leading RESTful web framework for Java applications is used to build RESTFul web services it has two part Restlet API and a Restlet implementation much like Servlet specification. There are many implementation of Restlet framework available you just need to add there jar in your classpath to use them. By using Restlet web framework you can write client and server.

Java JAX-RS also can be used to develop Restfull web services.

(<https://spring.io/guides/gs/rest-service/>)

REST in Spring 3: @MVC

<http://spring.io/blog/2009/03/08/rest-in-spring-3-mvc/>

**What is Resource in REST framework ?**

it represent a "resource" in REST architecture. on RESTLET API it has life cycle methods like **init(),** **handle()** and **release()** and contains a Context, Request and Response corresponding to specific target resource. This is now deprecated over ServerResource class and you should use that. see **Restlet** documentation for more details.

**5) Can you use Restlet without any web-container ?**

Yes, Restlet framework provide default server which can be used to handle service request in web container is not available.

**6) What is difference between Restlets and Jersey ?**

This *REST web service interview questions is open for you* all, post you answer in comment section.

**7) What is RESTEasy ?**

RESTEasy is another REST framework introduced in JBoss Application Server. This was rather easy REST interview questions. you can answer in detail only if you have used this or working in JBoss.

**8) What are the tools used for creating RESTFull web services ?**

You can use AJAX(Asynchronous JavaScript with XAML) and Direct Web Removing to consume web services in web application. Both Eclipse and NetBeans also supported development of RESTFul services.

**9) How to display custom error pages using RestFull web services ?**

In order to customize error you need to extend StatusService and implement getRepresentation(Status, Request, Response) method with your custom code now assign instance of your CustomStatusService to appropriate "statusService property".

**10)** **Which HTTP methods are supported by RestFull web services ?**

Another common REST interview questioning RESTFul web service each Resource supports **GET, POST, PUT and DELETE http methods**. GET is mapped to represent(), POST - acceptRepresentation(), PUT- storeRepresentation and DELET for rmeoveRepresentation.

**11) What is difference between top-down and bottom-up approach of developing web services ?**

In top-down approach first WSDL document is created and than Java classes are developed based on WSDL contract, so if WSDL contract changes you got to change your Java classes while in case of bottom up approach of web service development you first create Java code and then use annotations like @WebService to specify contract or interface and WSDL field will be automatically generated from your build.

**12) What happens if RestFull resources are accessed by multiple clients ? do you need to make it thread-safe?**

Since a new Resource instance is created for every incoming Request there is no need to make it thread-safe or add synchronization. multiple client can safely access RestFull resources concurrently.

In JAX-WS, a web service operation invocation is represented by an XML-based protocol such as SOAP. The SOAP specification defines the envelope structure, encoding rules, and conventions for representing web service invocations and responses. These calls and responses are transmitted as SOAP messages (XML files) over HTTP.

Although SOAP messages are complex, the JAX-WS API hides this complexity from the application developer. On the server side, the developer specifies the web service operations by defining methods in an interface written in the Java programming language. The developer also codes one or more classes that implement those methods. Client programs are also easy to code. A client creates a proxy (a local object representing the service) and then simply invokes methods on the proxy. **With JAX-WS, the developer does not generate or parse SOAP messages. It is the JAX-WS runtime system that converts the API calls and responses to and from SOAP messages.**

**Difference between URI and URL**

Uniform Resource Identifier (URI) is a string of characters used to identify a name or a resource on the Internet

An URI identifies a resource either by location, or a name, or both. **A URI has two specializations known as URL and URN.**

An Uniform Resource Locator (URL) is a subset of the Uniform Resource Identifier (URI) that specifies where an identified resource is available and the mechanism for retrieving it.URL defines how the resource can be obtained. It does not have to be HTTP URL (http://), a URL can also be (ftp://) or (smb://)

An Uniform Resource Name (URN) is a Uniform Resource Identifier (URI) that uses the URN scheme, and **does not imply availability of the identified resource**. Both URNs (names) and URLs (locators) are URIs, and a particular URI may be both a name and a locator at the same time.

The URNs are part of a larger Internet information architecture which is composed of URNs, URCs and URLs.

**bar.html is not a URN**. A URN is similar to a person's name, while a URL is like a street address. The URN defines something's identity, while the URL provides a location. Essentially, "what" vs. "where". A URN has to be of this form <URN> ::= "urn:" <NID> ":" <NSS> where <NID> is the Namespace Identifier, and <NSS> is the Namespace Specific String.

**To put it differently:**

* A URL is a URI that identifies a resource and also provides the means of locating the resource by describing the way to access it
* A URL is a URI
* A URI is not necessarily a URL

I'd say the only thing left to make it 100% clear would be to have an example of an URI that is not an URL. We can use the examples in the RFC3986:

REST WEB SERVICE

<http://www.mkyong.com/tutorials/jax-rs-tutorials/>

Java API for RESTful Web Services (**JAX-RS**), is a set of APIs to developer REST service. JAX-RS is part of the Java **EE6**, and make developers to develop REST web application easily

[**Jersey**](http://jersey.java.net/)**and [RESTEasy](http://www.jboss.org/resteasy" \t "_blank) are popular JAX-RS implementations.**

Also there are other frameworks such as **Restlet and Jersey**

# [Which implementation of Restuful Webservices to be used](http://stackoverflow.com/questions/22618729/which-implementation-of-restuful-webservices-to-be-used) ?

# <http://stackoverflow.com/questions/22618729/which-implementation-of-restuful-webservices-to-be-used/22618896?noredirect=1#22618896>

I'd go for Spring MVC version of REST, it supports Dependency Inversion(DI), Also spring supports many pluggable features like support for freemarker template, jaxb, xmlbeans etc., you will have plenty of options on return data.

Also, i wouldn't disagree on using Jersey or RESTEasy, they are very lightweight compared to spring. If you don't want to use spring, then either one of the above mentioned should work as well, there are no major pros and cons.

Difference between Spring MVC and REST Webservice

<http://stackoverflow.com/questions/5930795/difference-between-servlet-and-web-service>

A **web service** is a service that provides service methods to computers using either REST or SOAP as its protocol. There are several ways to implement a web service. The most simple way to write a web service would be to write a class and annotate it with the @WebService and @WebMethod annotations from javax.jws, and then launch it from a main-method with:

Endpoint.publish("http://localhost:8089/myservice", new MyWebService());

The result is that you can view the **WSDL** at the registered URL and if you have SoapUI or any other SOAP client you can also test and use your web service.

A **servlet** on the other hand is used to transport **HTTP** requests and resonses. It can be used to write a web application with JSPs and HTML, or to serve XML and JSON responses (as in a RESTful service) and of course also to receive and return SOAP messages. You can think of it as **one layer below web services**.

A more comprehensive and practical approach is to write a web service with a framework and to publish it on an application server or servlet container such as Tomcat or JBoss. In this case you would use a Servlet to handle the transport of the HTTP requests which transmit your SOAP or REST messages.

To write a web service with servlet technology you can for example use JAX-WS (e.g. for SOAP). In order to write RESTful services, you can either use JAX-RS (with the reference implementation being [**Jersey**](https://jersey.java.net/)), or alternatively you can use **Spring WebMVC**, but afaik that is not the main purpose of this framework and Jersey is considerably easier to use.

Regarding the second question: The @Controller annotation is a stereotype annotation that tells spring something about what your bean is supposed to do. What exactly this controller will return depends on the actual implementation of your methods, you can configure spring to return plain text, HTML, JSON, XML or whatever you want.

A note on the side, a class that is annotated with @Controller is not yet a servlet, it is simply a bean. How you use servlets depends mainly on the Framework that you use. For example, when you use Spring, the servlet job is done by

Springs DispatcherServlet which in turn forwards requests to the correct beans. If you use Tomcat, then you can directly write your own servlets by simply subclassing the javax.servlet.http.HttpServlet class and overwriting the necessary methods such as doGet which responds to HTTP GET requests from your browser.

MVC is about how the inner side of your app works.

REST is about how your app "talks" with other apps.

You can combine them.

A lot of modern frameworks actually are MVC based and make implementing REST web services easy: Ruby on Rails, Java Spring Framework with SpringMVC , Django, Backbone.js

# How to consume a web service

[**http://stackoverflow.com/questions/291847/simply-consuming-a-web-service-in-java**](http://stackoverflow.com/questions/291847/simply-consuming-a-web-service-in-java)

[**Specify the wsdl url in pom.xml and have the wsdl jar downloaded to local machine**](http://stackoverflow.com/questions/19448910/specify-the-wsdl-url-in-pom-xml-and-have-the-wsdl-jar-downloaded-to-local-machin)

[**http://stackoverflow.com/questions/19448910/specify-the-wsdl-url-in-pom-xml-and-have-the-wsdl-jar-downloaded-to-local-machin**](http://stackoverflow.com/questions/19448910/specify-the-wsdl-url-in-pom-xml-and-have-the-wsdl-jar-downloaded-to-local-machin)

**\\Stubs and Skeletons**  
RMI uses a standard mechanism (employed in RPC systems) for communicating with remote objects: stubs and skeletons. A stub for a remote object acts as a client's local representative or proxy for the remote object. The caller invokes a method on the local stub which is responsible for carrying out the method call on the remote object. In RMI, a stub for a remote object implements the same set of remote interfaces that a remote object implements.  
  
**When a stub's method is invoked, it does the following:**   
  
**1)**initiates a connection with the remote JVM containing the remote object,   
**2)**marshal (writes and transmits) the parameters to the remote JVM,   
**3)**waits for the result of the method invocation,   
**4)**unmarshals (reads) the return value or exception returned, and   
**5)**returns the value to the caller.   
  
The stub hides the serialization of parameters and the network-level communication in order to present a simple invocation mechanism to the caller.   
  
In the remote JVM, each remote object may have a corresponding skeleton (in Java 2 platform-only environments, skeletons are not required).   
  
The skeleton is resonsible for dispatching the call to the actual remote object implementation.   
  
**When a skeleton receives an incoming method invocation it does the following:**   
  
**1)**unmarshals (reads) the parameters for the remote method,   
**2)**invokes the method on the actual remote object implementation, and   
**3)**marshals (writes and transmits) the result (return value or exception) to the caller.   
  
In the Java 2 SDK, Standard Edition, v1.2 an additional stub protocol was introduced that eliminates the need for skeletons in Java 2 platform-only environments. Instead, generic code is used to carry out the duties performed by skeletons in JDK1.1. Stubs and skeletons are generated by the rmic compiler.   
  
Reference:   
<http://java.sun.com/j2se/1.5.0/docs/guide/rmi/spec/rmi-arch2.html>

<http://stackoverflow.com/questions/11566609/difference-between-jax-ws-axis2-cxf>

# [Difference between Jax-ws, axis2, cxf](http://stackoverflow.com/questions/11566609/difference-between-jax-ws-axis2-cxf)

All three can be used to create webservices in java. As of I know JAX-WS is a specification and Axis2 and CXF are implementations. But Java 1.6 has implementation of JAx-WS if I am not wrong. So one can use java 1.6 to develop JAX-WS web services without using Axis2, CXF? Then what is the use of Axis2, CXF? Thanks

The JAX-WS implementation built into the JDK really is just the basic soap stuff. If you need any of the more complex WS-\* things like WS-Security, WS-RM, WS-Policy, etc..., you need to use one of the alternatives like **CXF or Metro or Axis2.** It can also depend on what you are trying to integrate with. For example, CXF has top notch Spring support as well as very good OSGi support.

CXF also has other things besides just JAX-WS. It has a compliant JAX-RS implementation as well and supports exposing services as both REST and SOAP very well. Has a W3C compliant SOAP/JMS implementation if that type of things is required. Basically, lots of stuff not available from the in-jdk JAX-WS impl.

Also see:

[Difference between Apache CXF and Axis](http://stackoverflow.com/questions/1243247/difference-between-apache-cxf-and-axis/1245386#1245386)

In short.

WSDL WS-\* are language-agnostic.

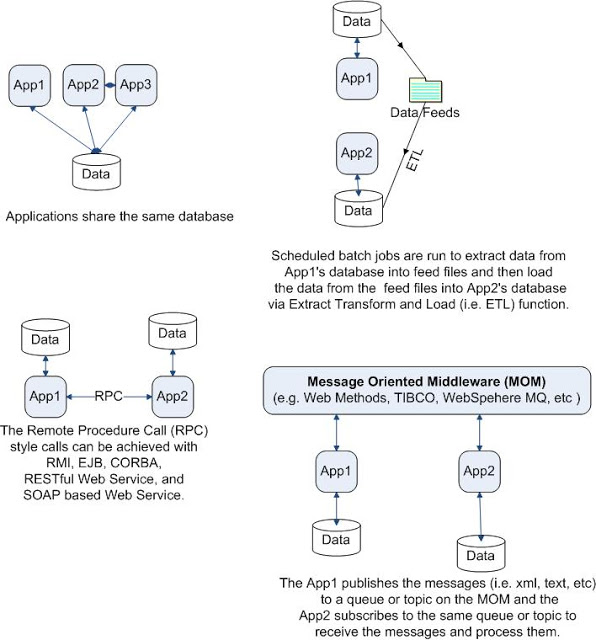
JAX-WS are Java standard to build web service.

[Apache CXF](http://cxf.apache.org/docs/why-cxf.html) and [Apache Axis 2](http://axis.apache.org/axis2/java/core/) are two implementations of JAX-WS. They also offer JAX-RS implementations so that you can build Restful services.

CXF has better integration with Spring, and Camel([camel-cxf](http://camel.apache.org/components.html)). And Axis 2 seems not have a active release.

### Java Web Services Interview Questions and Answers: Overview

**Q.** What are the different application integration styles?  
**A.**There are a number of different integration styles like  
  
1. Shared database  
2. batch file transfer  
3. Invoking remote procedures (RPC)  
4. Exchanging asynchronous messages over a message oriented middle-ware (MOM).

[](http://1.bp.blogspot.com/-PzRrLNGPU9U/TzSyk9vzpYI/AAAAAAAAASk/wsDCrNTg9FM/s1600/integration-styles.JPG)

**Q.**What are the different styles of Web Services used for application integration?   
**A.** **SOAP WS** and **RESTful**Web Service  
  
**Q.** What are the differences between both SOAP WS and RESTful WS?    
**A.**

|  |  |
| --- | --- |
|  | * The SOAP WS supports both remote procedure call (i.e. RPC) and message oriented middle-ware (MOM) integration styles. The Restful Web Service supports only RPC integration style. * The SOAP WS is transport protocol neutral. Supports multiple protocols like HTTP(S),  Messaging, TCP, UDP SMTP, etc. The REST is transport protocol specific. Supports only HTTP or HTTPS protocols. |

**The SOAP WS permits only XML data format.You define operations, which tunnels through the POST. The focus is on accessing the named operations and exposing the application logic as a service.** The REST permits multiple data formats like XML, JSON data, text, HTML, etc. Any browser can be used because the REST approach uses the standard GET, PUT, POST, and DELETE Weboperations. The focus is on accessing the named resources and exposing the data as a service. REST has AJAX support. It can use the XMLHttpRequest object. Good for stateless CRUD (Create, Read, Update, and Delete) operations.

**Q.** How would you decide what style of Web Service to use? SOAP WS or REST?  
**A**. In general, a REST based Web service is preferred due to its simplicity, performance, scalability, and support for multiple data formats. SOAP is favored where service requires comprehensive support for security and transactional reliability.

The answer really depends on the functional and non-functional requirements. Asking the questions listed below will help you choose.

* **Does the service expose data or business logic?**(REST is a better choice for exposing data, SOAP WS might be a better choice for logic).Do the consumers and the service providers require a formal contract? (SOAP has a formal contract via WSDL)
* **Do we need to support multiple data formats?**
* **Do we need to make AJAX calls?**(REST can use the XMLHttpRequest)
* **Is the call synchronous or  asynchronous?**
* **Is the call stateful or stateless?** (REST is suited for statless CRUD operations)
* **What level of security is required?** (SOAP WS has better support for security)
* **What level of transaction support is required?**(SOAP WS has better support for transaction management)
* **Do we have limited band width?** (SOAP is more verbose)
* **What’s best for the developers who will build clients for the service?**(REST is easier to implement, test, and maintain)

What tools do you use to test your Web Services?  
**A.** **SoapUI** tool for SOAP WS and the Firefox "**poster**" plugin for RESTFul services.

**Q.** What is the difference between SOA and a Web service?   
**A.**  
  
**SOA is**a software design principle and an architectural pattern for implementing loosely coupled, reusable and coarse grained services. You can implement SOA using any protocols such as HTTP, HTTPS, JMS, SMTP, RMI, IIOP (i.e. EJB uses IIOP), RPC etc. Messages can be in XML or Data Transfer Objects (DTOs).       
  
**Web service is** an implementation technology and one of the ways to implement SOA. You can build SOA based applications without using Web services – for example by using other traditional technologies like Java RMI, EJB, JMS based messaging, etc. But what Web services offer is the standards based  and platform-independent service via HTTP, XML, SOAP, WSDL and UDDI, thus allowing interoperability between heterogeneous technologies such as J2EE and .NET.

<https://www.udemy.com/blog/web-services-interview-questions/>

## What are the steps to get a proxy object of a web service at the client side?

There are three steps to get a proxy object of a web service at the client side.

1. Access UDDI node for a list of web services.

2. Services thus responded by UDDI have URL pointing to DISCO or WSDL document.

3. Parse DISCO and WSDL document and build a proxy object which can communicate with the web service.

## What types of operations are available in WSDL?

There are four operations available:

1. One-way, where the operation can receive a message but will not return a response.

2. Request-response, where the operation can receive a request and will return a response.

3. Solicit-response, where the operation can send a request and will wait for a response.

4. Notification, where the operation can send a message but will not wait for a response

## Define a REST web service.

REST is Representational State Transfer and it is a network of web pages where the client progresses through an application by selecting links. REST is an architectural style that uses existing standards such as HTTP.

**What is the difference between HTTP POST and PUT requests in REST?**  
**POST is used to create where as PUT is used to (update , if does not exist create it)/ Replace.**

<http://www.techsutras.com/2014/04/examples-of-rest-web-service.html>

web services

### 1. Twitter API

The Twitter micro-blogging service includes two RESTful APIs.  
The Twitter REST API methods allow developers to access core Twitter data like timelines, status data, and user information.  
[API Documentation](https://dev.twitter.com/docs/api/1.1)  
  
Sample Resource description  
**GET statuses/retweets/:id**   Returns a collection of the 100 most recent retweets of the tweet specified by the id parameter.  
**GET search/tweets**  Returns a collection of relevant Tweets matching a specified query  
**GET followers/ids** Returns a cursored collection of user IDs for every user following the specified user.  
**GET users/suggestions** Access to Twitter's suggested user list.

### 2. Last FM API

The Last.fm API allows anyone to build their own programs using Last.fm data, whether they're on the web, the desktop or mobile devices.The RESTful API allows for access to the last.fm music data resources - albums, artists, playlists etc.  
[Last FM REST API](http://www.last.fm/api/rest)  
[Last FM API Intro](http://www.last.fm/api/intro)

### 3. LinkedIn API

LinkedIn is the worlds largest business social networking hub.  
The REST API provides a simple, consistent representation of people, companies,jobs, and the interactions and relationships between them.  
[LinkedIn REST API Documentation](https://developer.linkedin.com/rest)  
  
Profile API  
GET [http://api.linkedin.com/v1/people/id=](https://developer.linkedin.com/documents/profile-api)<MemberId>  
  
Group API  
GET [http://api.linkedin.com/v1/groups/<GroupId](http://api.linkedin.com/v1/groups/%3cGroupId)>

***JSON Vs XML, Which one to use ?***  
Following are key differences between JSON & XML, which will help to decide which content format one should use in there application :

* ***Parsing and Generating***JSON data is easier as compared to XML.
* JSON has ***simpler structure***than XML.
* In XML ***document manipulation*** is easy as compared to JSON.You can easily find/update nodes.You have XPath/XQuery for XML, Do we have it for JSON ?
* You can do lots of thing with XML for example ***converting XML to HTML using XLST,*** such tools not available for JSON.
* XML is more ***rich in features*** and you can have ***more control over data*** and do great amount of validation as compared to JSON.You can have DTD's for XML to specify validation in details
* ***Working with JSON is simpler***, especially with JavaScript, just by giving call to Eval will give you JavaScript Object.
* JSON is not in a ***document format neither it is markup language***but XML is.
* JSON has ***great language support*** as compared to XML.
* JSON is best for***simpler use/scenario***, for better security, support and validation XML is better.
* JSON is ***faster*** as compared to XML as it is less rich, there is less overhead of tags & there is simpler validation rules

EXAMPLES”:

* 1. soap

<http://crunchify.com/create-sample-wsdl-in-eclipse-and-generate-client/>

[CRUNCHIFY](http://crunchify.com/)

* [Home](http://crunchify.com/)

* [Deals](http://deals.crunchify.com/)

* [Java](http://crunchify.com/category/java-web-development-tutorial/)

* [Spring MVC](http://crunchify.com/category/spring-mvc/)

* [Tutorials ↡](http://crunchify.com/category/my-collection/)
  + [·  Eclipse Tutorials](http://crunchify.com/category/eclipse/)
  + [·  Apache Tomcat](http://crunchify.com/category/apache-tomcat/)
  + [·  jQuery Articles](http://crunchify.com/category/jquery/)
  + [·  JSON Articles](http://crunchify.com/category/json/)
  + [·  AJAX Articles](http://crunchify.com/category/ajax-2/)
  + [·  Mac Tips](http://crunchify.com/category/mac/)
  + [·  JavaScript Articles](http://crunchify.com/category/java-script/)
  + [·  HTML5 Articles](http://crunchify.com/category/html5/)
  + [·  Social Media News](http://crunchify.com/category/social-media/)

* [All in One Webmaster](http://crunchify.com/all-in-one-webmaster/)

* [WordPress ↡](http://crunchify.com/category/wordpress/)
  + [·  Consultant you trust](http://crunchify.com/services/)
  + [·  Facebook Members](http://crunchify.com/facebook-members/)
  + [·  Foursquare Checkins](http://crunchify.com/foursquare-checkins/)
  + [·  WordPress Plugins](http://crunchify.com/category/plugins/)
  + [·  WordPress Themes](http://crunchify.com/category/themes/)

* [Ads](http://crunchify.com/advertise/)

* [Contact ↡](http://crunchify.com/contact/)
  + [·  Disclaimer](http://crunchify.com/disclaimer/)

We help customers create an effective online presence...

[Crunchify](http://crunchify.com/)  ➢  [Eclipse](http://crunchify.com/category/eclipse/)  ➢  How to Create Sample WSDL in Eclipse and Generate Client

# How to Create Sample WSDL in Eclipse and Generate Client

Last Updated on *December 17th, 2014* by [Crunchify](http://crunchify.com/about) [10 Comments](http://crunchify.com/create-sample-wsdl-in-eclipse-and-generate-client/#disqus_thread)

Have you ever tried creating Simple [Java](http://crunchify.com/category/java-web-development-tutorial/) Web Service Definition Language in [Eclipse](http://crunchify.com/category/eclipse/)? Well, here are few simple steps to create WSDL in Eclipse environment and Generate/Test Client.

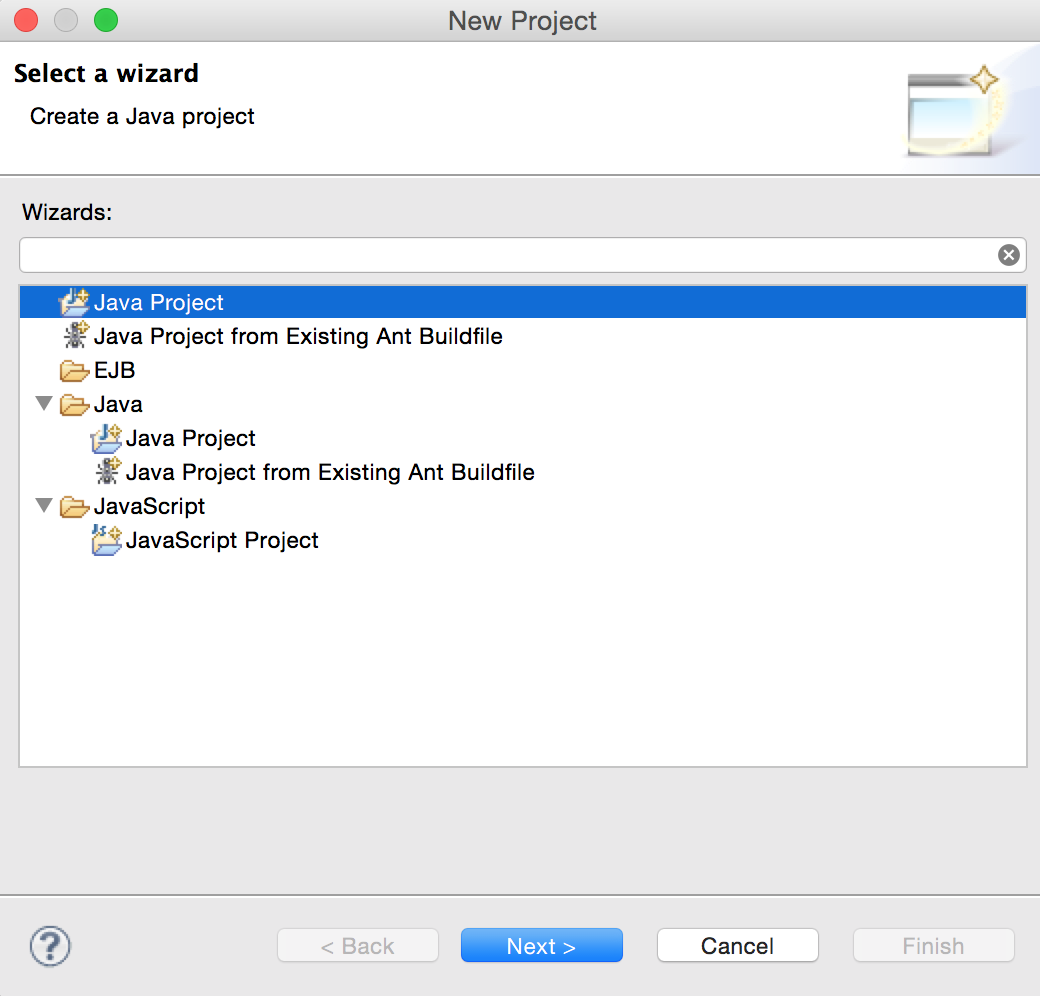
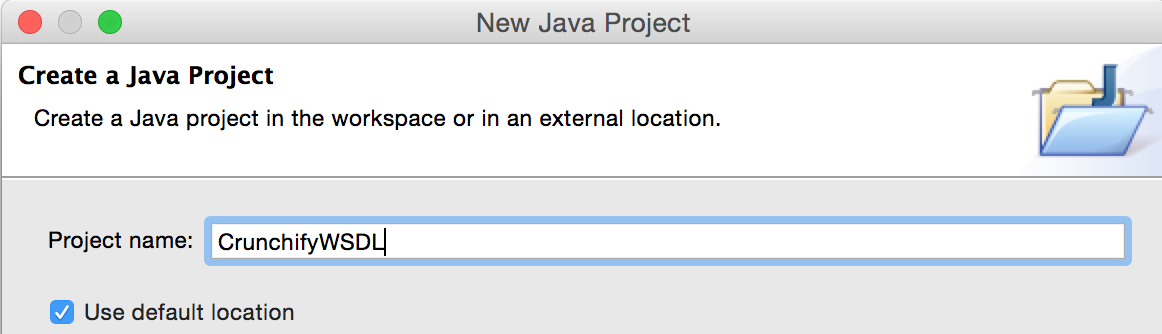
Hope you find it useful. Also, if you have any of below questions then you are right location.

1. Generating a client from WSDL – Eclipse
2. Generating an Apache Axis2 Java client proxy from a WSDL
3. Generating a Java client proxy and a sample application
4. Generating Web Service Client
5. generate client from wsdl using axis2 eclipse

### Let’s get started:

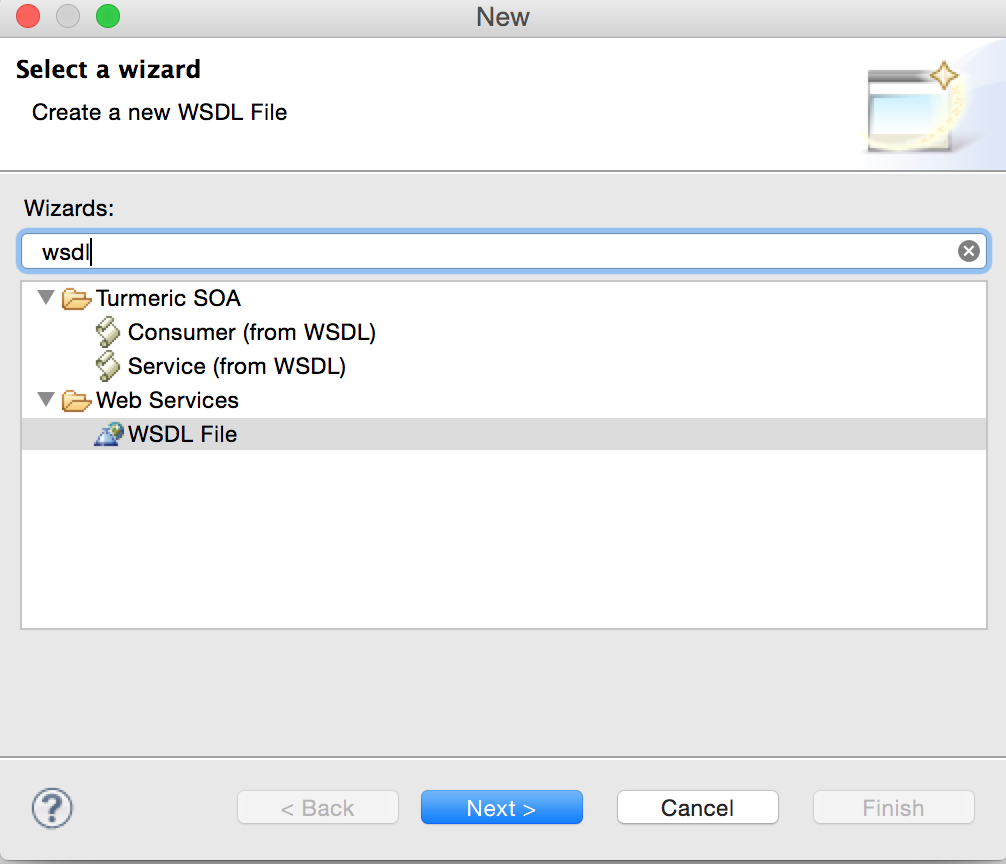
#### Step-1

Create Java Project Called “CrunchifyWSDL“. File -> New Project -> Java Project -> Provide Name -> Finish.

[](http://cdn2.crunchify.com/wp-content/uploads/2012/10/Crunchify-Create-Java-Project.png)[](http://cdn1.crunchify.com/wp-content/uploads/2012/10/Java-Project-CrunchifyWSDL.png)

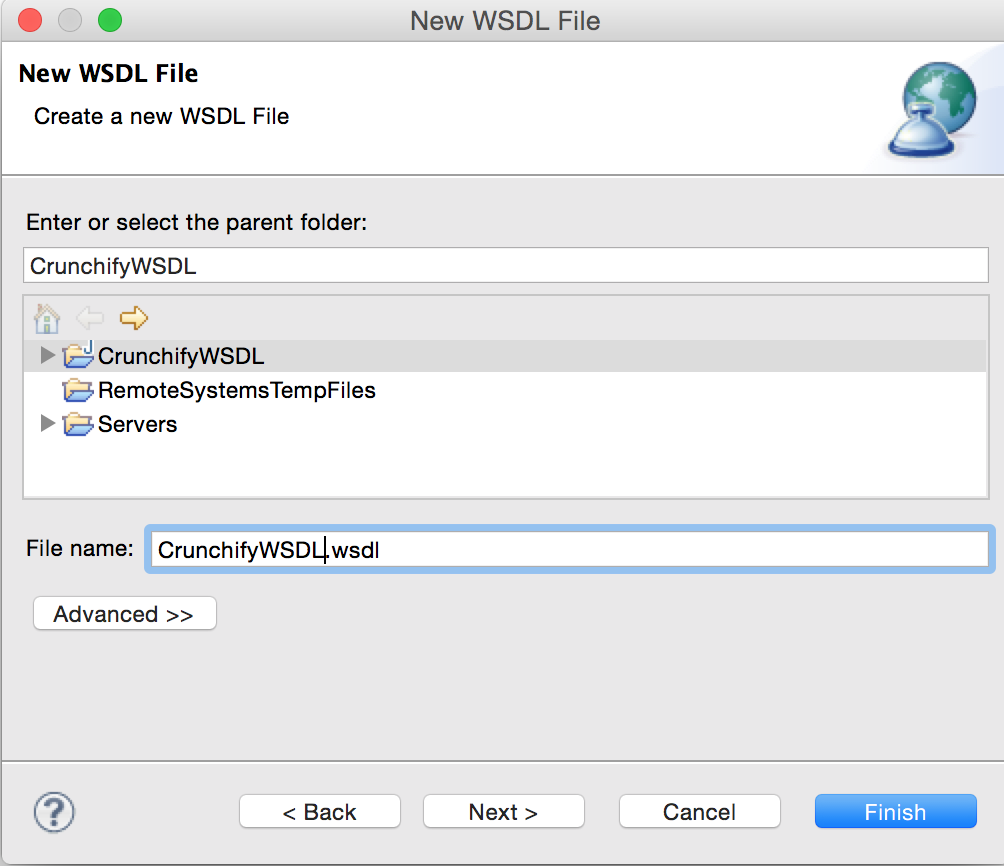
#### ****Step 2.****

In the workbench, click File > New > Other and select Web Services > WSDL. Click Next.

[](http://cdn2.crunchify.com/wp-content/uploads/2012/10/Eclipse-Create-New-WSDL-file.png)

#### ****Step 3.****

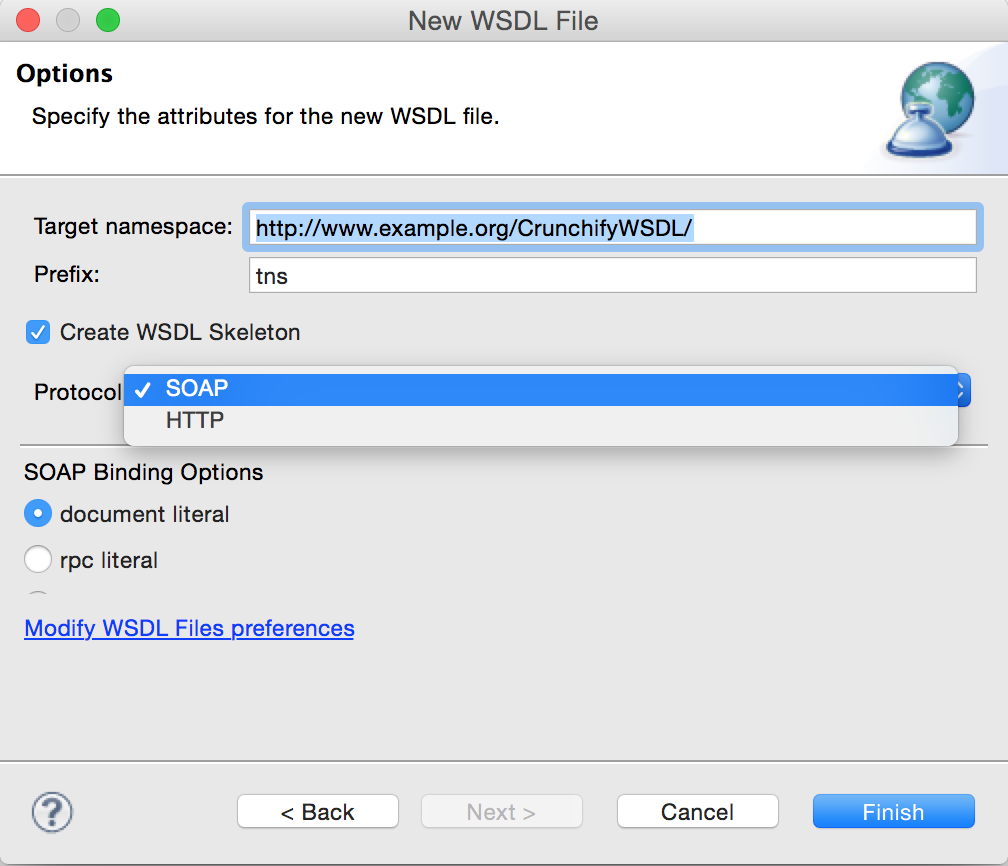
Select the project CrunchifyWSDL that will contain the WSDL file. In the File name field, type the name of the WSDL file, i.e. CrunchifyWSDL.wsdl. The name of your XML file must end in .wsdl.

[](http://cdn3.crunchify.com/wp-content/uploads/2012/10/Create-New-WSDL-file-CrunchifyWSDL.png)

#### ****Step 4.****

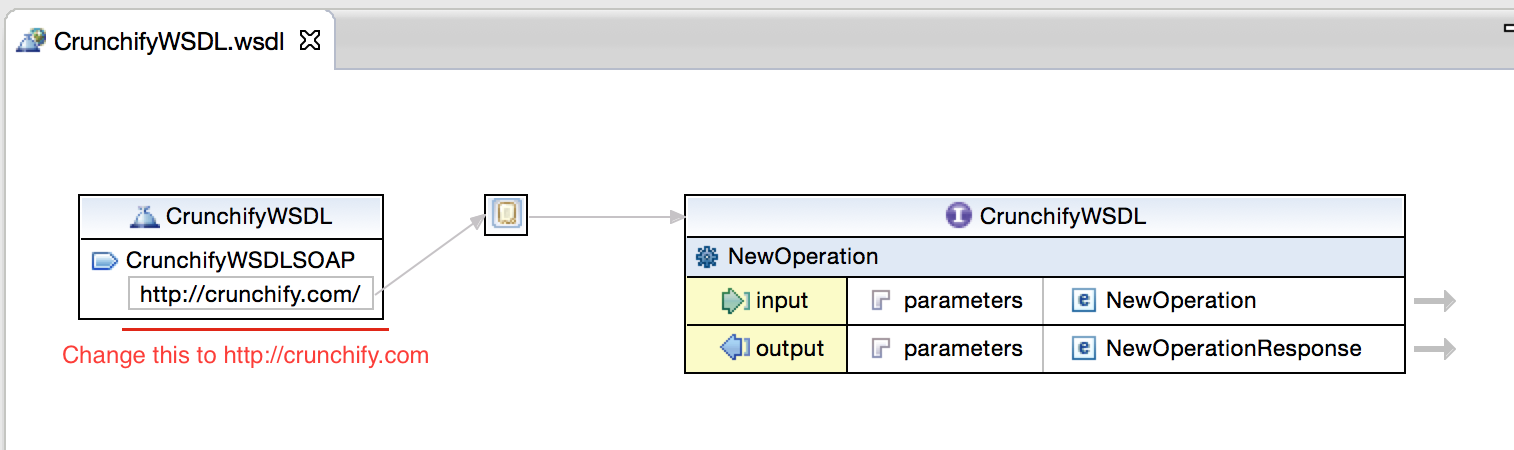
Click Next and enter the following information on the next page of the wizard:

1. A Target namespace for the WSDL file or accept the default (http://www.example.org/MyWSDLFile/). The target namespace is used for the names of messages and the port type, binding and service defined in the WSDL file. The value must take the form of a valid URI (for example, http://www.mycompany.com/myservice/)
2. The Prefix associated with the target namespace.
3. Select Create WSDL Skeleton if you want the wizard to create the skeleton of the WSDL file. This will generate the WSDL elements required for your service, including bindings, ports and messages. You can then modify these to meet the requirements of your Web service .
4. If you have chosen to create a WSDL skeleton, select the binding options you want to use in the Protocol drop down. The options are SOAP and HTTP. Use the SOAP protocol when you want to exchange structured and typed information. Use the HTTP protocol when you want your application client to just request or update information.
5. If you select SOAP you can then select the encoding style you want to use:
   * document literal. Document style messages, literal encoding. Use this style of binding when you want to send SOAP messages that can be validated by an XML validator. All the data types in the [SOAP message](http://crunchify.com/soap-vs-rest-simple-object-access-protocol-vs-representational-state-transfer/) body are defined in a schema, so the WSDL parts must point to schema elements.
   * rpc literal. RPC style messages, literal encoding. Use this style of binding when you want to specify the operation method names in your SOAP messages so a server can dispatch the specified methods. Data types must be defined, so the WSDL parts must point to XSD types.
   * rpc encoded. RPC style messages and SOAP encoding. Use this style of binding when you want to encode data graphs in your SOAP messages so a server can deserialize the object data. Data types must be defined, so the WSDL parts must point to XSD types.
6. If you select HTTP you can select whether to create an HTTP getter or setter.
   * [HTTP GET](http://crunchify.com/simple-way-to-get-http-response-header-in-java/). A GET request fetches data from a Web server based on an URL value and a set of HTTP headers. Use this method when you want to retrieve information specified in the request.
   * HTTP POST. A POST request sends additional data to the server, specified after the URL and the headers. Use this method when you want to send data enclosed in the body of the request.

[](http://cdn1.crunchify.com/wp-content/uploads/2012/10/Specify-Attributes-for-New-WSDL-file.png)

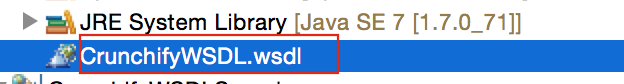
#### ****Step 5.****

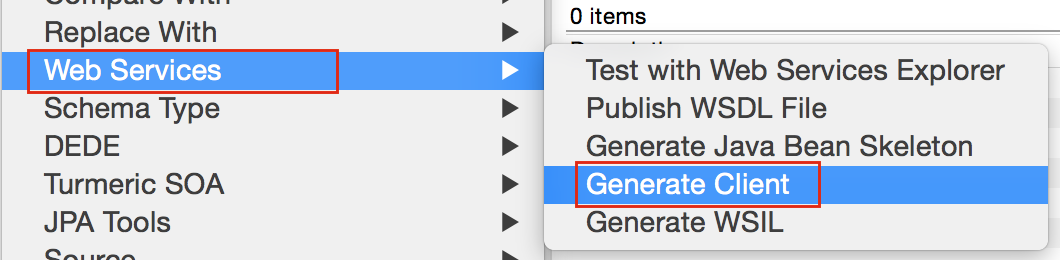
Open newly generated wsdl and change endpoint for testing as mentioned below.

[](http://cdn2.crunchify.com/wp-content/uploads/2012/10/CrunchifyWSDL-SOAP-End-point-change-in-Eclipse.png)

#### ****Step 6.****

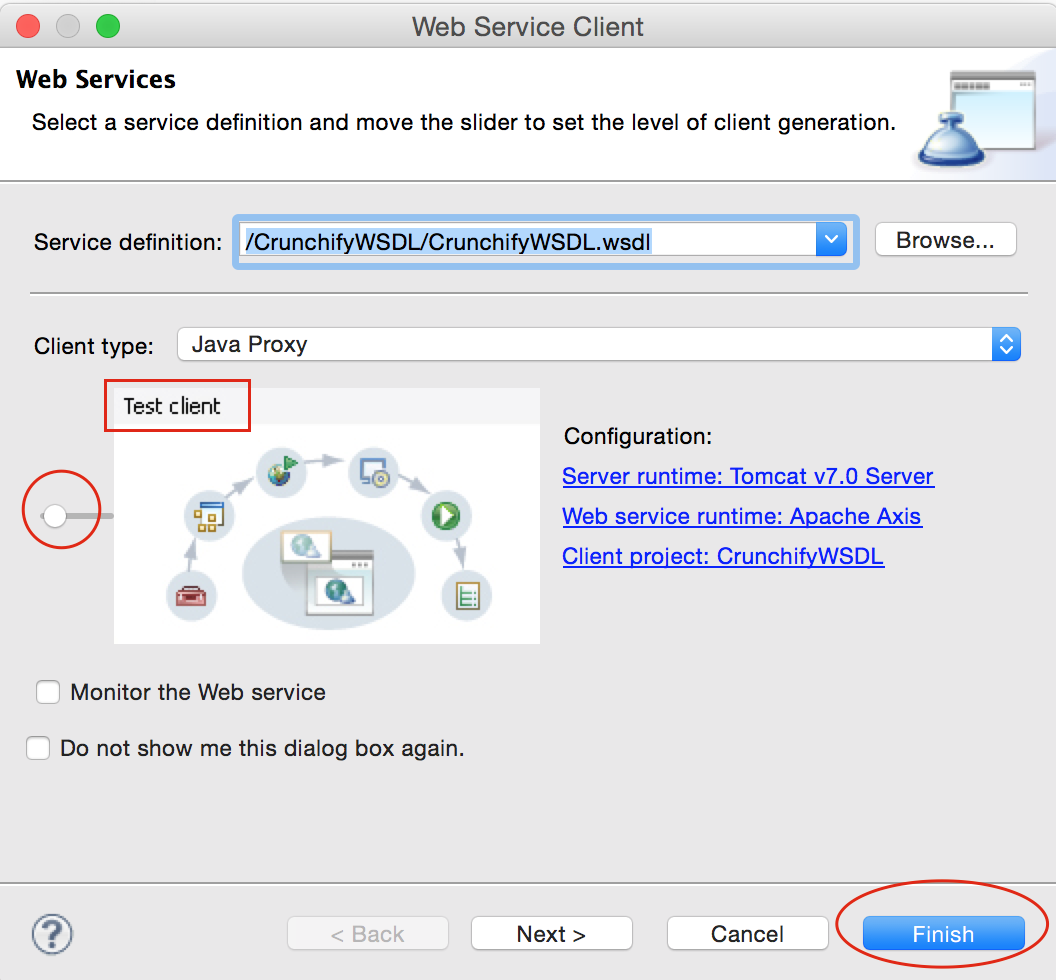
Right Click on CrunchifyWSDL.wsdl -> Web Services -> Generate Client

[](http://cdn3.crunchify.com/wp-content/uploads/2012/10/Right-click-on-wsdl.png)

[](http://cdn2.crunchify.com/wp-content/uploads/2012/10/Generate-Client-WSDL.png)

#### ****Step 7.****

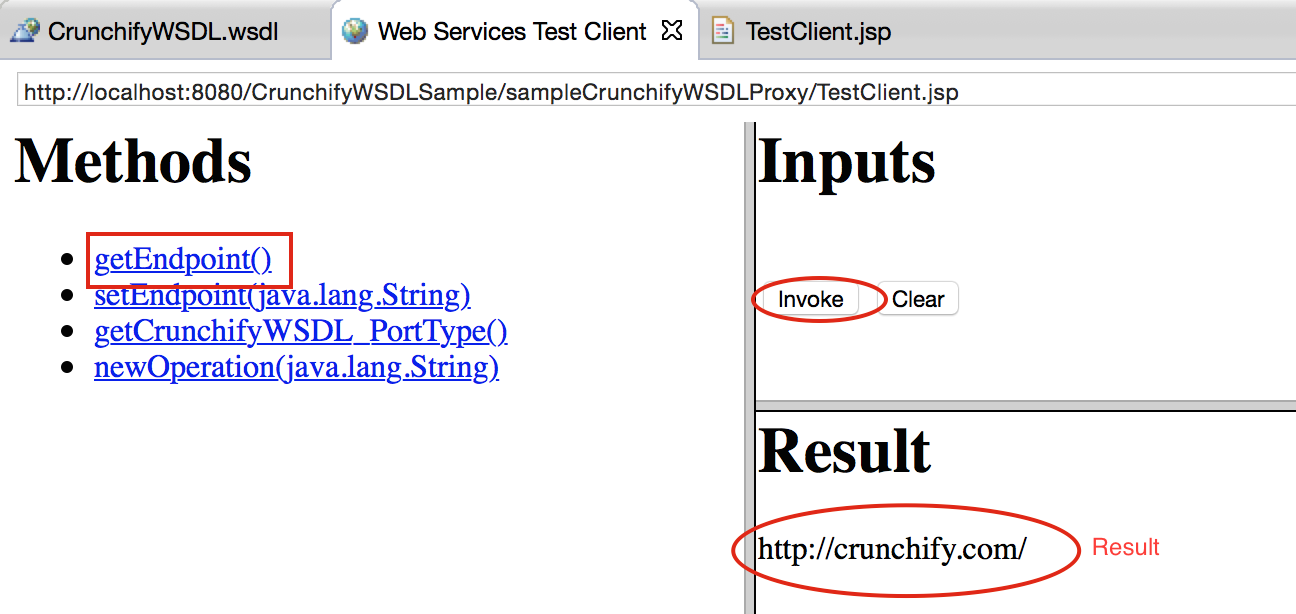
Select "Test Client" -> Finish.

[](http://cdn.crunchify.com/wp-content/uploads/2012/10/Generate-Client-Test-SOAP-in-Eclipse.png)

If you see “org.eclipse.jst.ws.util.JspUtils cannot be resolved to a type” Error in Eclipse then follow these steps: <http://crunchify.com/how-to-fix-org-eclipse-jst-ws-util-jsputils-cannot-be-resolved-to-a-type-error-in-eclipse/>

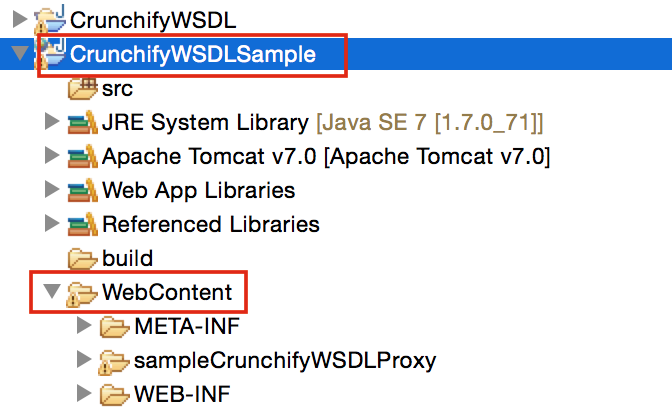
#### ****Step 8.****

Click on "getEndpoint()" -> Invoke -> See result.

[](http://cdn2.crunchify.com/wp-content/uploads/2012/10/WSDL-Invoke-Operation.png)

#### Other points to note:

You will see new project “CrunchifyWSDLSample” created in Eclipse.

[](http://cdn.crunchify.com/wp-content/uploads/2012/10/CrunchifySampleWSDL.png)

Have anything to add to this article? Please chime in and join the conversion.

[](http://twitter.com/share?url=http://crunchify.com/create-sample-wsdl-in-eclipse-and-generate-client/&text=How+to+Create+Sample+WSDL+in+Eclipse+and+Generate+Client+via+@Crunchify)[](http://www.facebook.com/sharer.php?u=http://crunchify.com/create-sample-wsdl-in-eclipse-and-generate-client/)[](https://plus.google.com/share?url=http://crunchify.com/create-sample-wsdl-in-eclipse-and-generate-client/)[](javascript:void((function()%7bvar%20e=document.createElement('script');e.setAttribute('type','text/javascript');e.setAttribute('charset','UTF-8');e.setAttribute('src','//assets.pinterest.com/js/pinmarklet.js?r='+Math.random()*99999999);document.body.appendChild(e)%7d)());)[](http://www.tumblr.com/share/link?url=crunchify.com/create-sample-wsdl-in-eclipse-and-generate-client/&name=How%20to%20Create%20Sample%20WSDL%20in%20Eclipse%20and%20Generate%20Client)[](https://bufferapp.com/add?url=http://crunchify.com/create-sample-wsdl-in-eclipse-and-generate-client/&text=%20How%20to%20Create%20Sample%20WSDL%20in%20Eclipse%20and%20Generate%20Client)

### Some more articles you might also be interested in …

1. [Introduction to WSDL (Web Service Definition Language) – Sample HelloWorld Tutorial](http://crunchify.com/web-service-definition-language-sample-helloworld-wsdl/)
2. [Basic WSDL Structure Understanding – (Web Service Description Language) Explained](http://crunchify.com/basic-wsdl-structure-understanding-wsdl-explained/)
3. [SOAP vs REST: Simple Object Access Protocol Vs. REpresentational State Transfer](http://crunchify.com/soap-vs-rest-simple-object-access-protocol-vs-representational-state-transfer/)
4. [How to Create Dynamic Web Project using Maven in Eclipse?](http://crunchify.com/how-to-create-dynamic-web-project-using-maven-in-eclipse/)
5. [Eclipse: Missing web.xml file? How can I create web.xml in Eclipse?](http://crunchify.com/eclipse-missing-web-xml-file-how-can-i-create-web-xml-in-eclipse/)
6. [How to Configure BitBucket Git Repository in your Eclipse Environment? Git version control with Eclipse (EGit)](http://crunchify.com/how-to-configure-bitbucket-git-repository-in-you-eclipse/)
7. [How to Create RESTful Java Client With Jersey Client – Example](http://crunchify.com/how-to-create-restful-java-client-with-jersey-client-example/)
8. [How to Create a .war file from Eclipse using Maven Plugin? Apache ‘maven-war-plugin’ usage](http://crunchify.com/how-to-create-a-war-file-from-eclipse-using-maven-plugin-apache-maven-war-plugin-usage/)
9. [How to exclude JSP file validation in Eclipse?](http://crunchify.com/how-to-exclude-jsp-file-validation-in-eclipse/)
10. [How to solve “Dynamic Web Module 3.1 requires Java 1.7 or newer” in Eclipse](http://crunchify.com/how-to-solve-dynamic-web-module-3-1-requires-java-1-7-or-newer-in-eclipse/)

Filed Under: [Eclipse](http://crunchify.com/category/eclipse/), [Java Collection](http://crunchify.com/category/java-web-development-tutorial/)Tagged: [eclipse](http://crunchify.com/tag/eclipse-environment/), [generate client from wsdl file](http://crunchify.com/tag/generate-client-from-wsdl-file/), [Generating a client from WSDL - Eclipse](http://crunchify.com/tag/generating-a-client-from-wsdl-eclipse/), [Java Eclipse SOAP tutorial](http://crunchify.com/tag/java-eclipse-soap-tutorial/), [soap](http://crunchify.com/tag/soap/),[Web Service Definition Language](http://crunchify.com/tag/web-service-definition-language/), [WSDL](http://crunchify.com/tag/wsdl/), [WSDL Java Crunchify Tutorial](http://crunchify.com/tag/wsdl-java-crunchify-tutorial/)

#### Enjoyed this post?

Be sure to subscribe to the Crunchify newsletter and get regular updates about awesome posts just like this one and more! Join more than 8000 subscribers!

Top of Form

Enter your email address... 

Bottom of Form



#### About Crunchify

I’m [Arpit](http://crunchify.com/about) from NYC, the founder of Crunchify. To make a long story very short, I love [Web Development](http://crunchify.com/category/java-web-development-tutorial/), help customers get the most out of their [WordPress](http://crunchify.com/category/wordpress/) sites with an effective Online presence. Master with Genesis Framework. Love writing Java, Spring, J2EE, Web Technologies coding tutorials. Love to hear your thoughts and opinions on my articles directly.

[RSS Feed](http://feeds.feedburner.com/Crunchify)[Google+](https://www.google.com/+CrunchifyDotCom)[Facebook](https://www.facebook.com/Crunchify)[Twitter](http://twitter.com/Crunchify)[WordPress.org](http://profiles.wordpress.org/crunchify/)

Top of Form

Bottom of Form

* [Simplest Spring MVC Hello World Tutorial](http://crunchify.com/simplest-spring-mvc-hello-world-example-tutorial-spring-model-view-controller-tips/)
* [Noticed NullPointerException (NPE)? Try Best Practices to avoid NPE at Runtime](http://crunchify.com/have-you-noticed-nullpointerexception-npe-there-are-few-best-practices-to-avoid-npe/)
* [How to Start Embedded HTTP Jersey server during Java Application Startup](http://crunchify.com/how-to-start-embedded-http-jersey-server-during-java-application-startup/)
* [Java Properties File: How to Read config.properties Values in Java?](http://crunchify.com/java-properties-file-how-to-read-config-properties-values-in-java/)
* [Create Very Simple Jersey REST Service and Send JSON Data From Java Client](http://crunchify.com/create-very-simple-jersey-rest-service-and-send-json-data-from-java-client/)
* [How to build RESTful Service with Java using JAX-RS and Jersey (Example)](http://crunchify.com/how-to-build-restful-service-with-java-using-jax-rs-and-jersey/)
* [How to Refresh DIV Content Without Reloading Page using jQuery?](http://crunchify.com/how-to-refresh-div-content-without-reloading-page-using-jquery-and-ajax/)

**Top Posts**

* [Spring MVC Tutorials](http://crunchify.com/category/spring-mvc/)
* [300+ Java Study Tutorials](http://crunchify.com/category/java-web-development-tutorial/)
* [Apache Tomcat Tips & Tricks](http://crunchify.com/category/apache-tomcat/)
* [jQuery Tutorials](http://crunchify.com/category/jquery/)
* [JSON Tips](http://crunchify.com/category/json/)
* [250+ WordPress Hacks & Tips](http://crunchify.com/category/wordpress/)

**Tutorials**

[↞](http://crunchify.com/web-service-definition-language-sample-helloworld-wsdl/)

[↠](http://crunchify.com/eclipse-missing-web-xml-file-how-can-i-create-web-xml-in-eclipse/)

#### Recently Answered…

* [Arpit](http://crunchify.com/) on [How to Create a Simple In Memory Cache in Java (Lightweight Cache)](http://crunchify.com/how-to-create-a-simple-in-memory-cache-in-java-lightweight-cache/#comment-55708)
* [Arpit](http://crunchify.com/) on [Create Very Simple Jersey REST Service and Send JSON Data From Java Client](http://crunchify.com/create-very-simple-jersey-rest-service-and-send-json-data-from-java-client/#comment-55707)
* [Arpit](http://crunchify.com/) on [How to Create Sample WSDL in Eclipse and Generate Client](http://crunchify.com/create-sample-wsdl-in-eclipse-and-generate-client/#comment-55706)
* [Arpit](http://crunchify.com/) on [JVM Tuning: Heapsize, Stacksize & Garbage Collection Fundamental](http://crunchify.com/jvm-tuning-heapsize-stacksize-garbage-collection-fundamental/#comment-55705)
* [Arpit](http://crunchify.com/) on [Java XML XPath Parser – How to Parse XML Document using XPath in Java?](http://crunchify.com/java-xml-xpath-parser-how-to-parse-xml-document-using-xpath-in-java/#comment-55704)

#### Recently Crunched…

* [Working on Spring MVC Project? How to Report List of All Loaded Spring Beans during Startup?](http://crunchify.com/working-on-spring-mvc-project-how-to-report-list-of-all-loaded-spring-beans-during-startup/)
* [How to Setup PHP Development Tools (PDT) in Eclipse For WordPress Plugin Development](http://crunchify.com/how-to-setup-php-development-tools-pdt-in-eclipse-for-wordpress-plugin-development/)
* [In Java What is a Difference Between IdentityHashMap and HashMap + Performance Comparision](http://crunchify.com/in-java-what-is-a-difference-between-identityhashmap-and-hashmap-performance-comparision/)
* [How to Create a .war file from Eclipse using Maven Plugin? Apache ‘maven-war-plugin’ usage](http://crunchify.com/how-to-create-a-war-file-from-eclipse-using-maven-plugin-apache-maven-war-plugin-usage/)
* [How-To: Java Regex – Matcher (java.util.regex.Matcher) Tutorial](http://crunchify.com/java-regular-expression-matcher-all-in-one-tutorial/)

#### Archives



#### Search your Favorite Topics…

Top of Form

Bottom of Form

© Copyright 2012-14 **Crunchify** · All Rights Reserved · Delivered by [MaxCDN](http://w.crunchify.com/MaxCDN-Deals" \t "_blank) · Runs on the [Genesis](http://w.crunchify.com/GenesisFramework-Deals).   
[About](http://crunchify.com/about/) · [Archives](http://crunchify.com/sitemap/) · [Advertise](http://crunchify.com/advertise/) · Protected by [http://cdn.crunchify.com/wp-content/icon/dmca.png](http://www.dmca.com/)   
[↟](http://crunchify.com/create-sample-wsdl-in-eclipse-and-generate-client/)

REST FULL SERVICE USING JERSEY

<http://www.java2blog.com/2013/04/create-restful-web-servicesjax-rs-using.html>

Java API for RESTful Web Services (**JAX-RS**), is a set if APIs to developer REST service. JAX-RS is part of the Java EE6, and make developers to develop REST web application easily.

Jersey is the reference implementation for this specification. Jersey contains basically a REST server and a REST client. The core client can communicate with the server using jersey lib.  
  
On the server side Jersey uses a servlet which scans predefined classes to identify RESTful resources. Via the web.xml configuration file for your web application.  
  
The base URL of this servlet is:

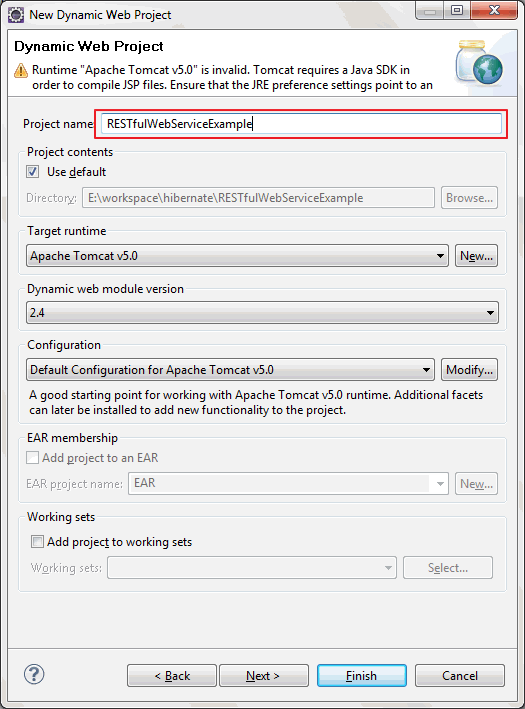
**http://your\_domain:port/display-name/url-pattern/path\_from\_rest\_class**

This servlet analyzes the incoming HTTP request and selects the correct class and method depending on  request. This selection is based on annotations provided in the class and methods.

#### Prerequisites:

* Java SE 6
* Download the zip of Jersey files from this location – <http://jersey.java.net/nonav/documentation/latest/chapter_deps.html>
* Eclipse IDE

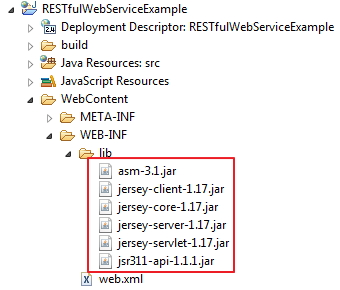
1) Open eclipse.  
2) Create new dynamic web project named "RESTfulWebServiceExample"

[](http://4.bp.blogspot.com/-AAzM2Wo2EBw/UV0zq-hdD5I/AAAAAAAAA8M/GEBLNsPKMy8/s1600/NewProjectRESTWSExample.gif)

3) Now go to location where you have download jersey and go to jersey-archive-1.17->lib  
folder.you can have all jars but for now you can copy following jars

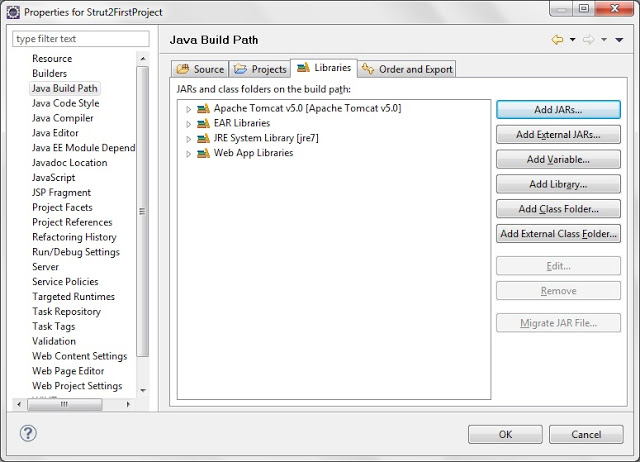
* asm-3.1
* jersey-client-1.17
* jersey-core-1.17
* jersey-server-1.17
* jersey-servlet-1.17
* jsr311-api-1.1.1

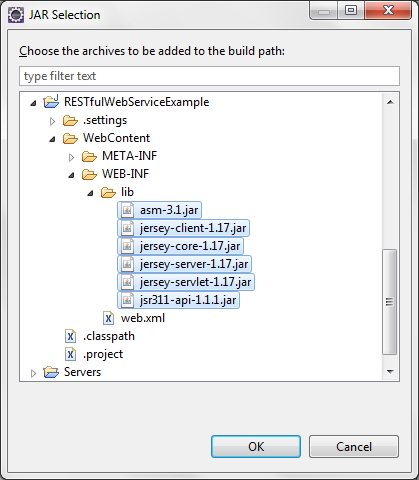
Paste all above copied jars to WebContent->WEB-INF->lib

[](http://1.bp.blogspot.com/-2aUfeDX0qeM/UV1T9XRSzRI/AAAAAAAAA8s/BtkoMGqUXcI/s1600/jerseyJarsPastedToLib.jpg)

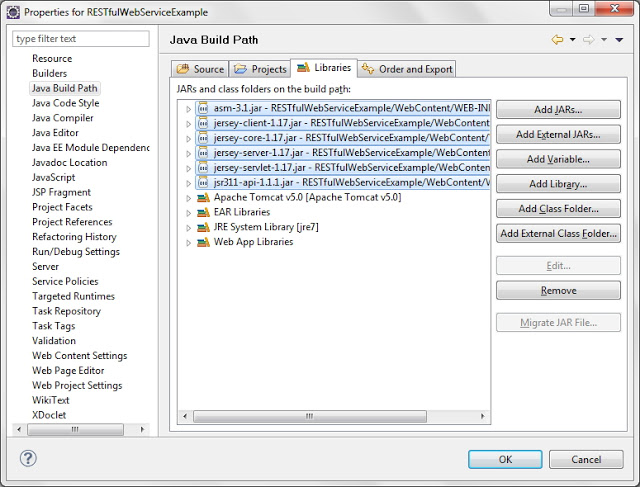
Add all these jars to eclipse build path.

Right click on project(RESTfulWebServiceExample)->properties

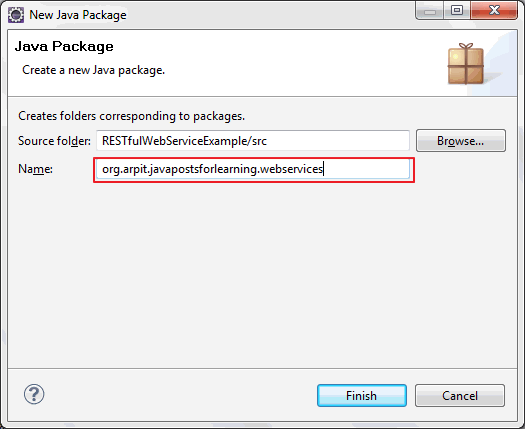
[](http://4.bp.blogspot.com/-K4toq5H1YyE/UEjTWjZpxVI/AAAAAAAAAUM/rkmajZPmsw8/s1600/javaBuildPath.jpg)  
Click on Java Build Path and then Add jars as shown in above diagram.

[](http://4.bp.blogspot.com/-6YvxoHkXXSk/UV1VIPLY36I/AAAAAAAAA84/6R3Q2V1mpcY/s1600/jerseyJarsAdded.jpg)

go to project->WebContent->WEB-INF->lib and select all jars then click on ok.

[](http://3.bp.blogspot.com/-lQulzeG7NDI/UV1Vc6rWEVI/AAAAAAAAA9A/D2q2KozSNfM/s1600/jerseyJarsCompletelyAdded.jpg)

Click ok.Jersey jars added to class path.   
4) Create new package named "org.arpit.javapostsforlearning.webservice"

[](http://1.bp.blogspot.com/-nqe3eq1w258/UV0zzJZ9tjI/AAAAAAAAA8U/Q6GNCjkwbRc/s1600/NewPackageRESTWSExample.gif)

5)Create  **FeetToInchAndInchToFeetConversionService.java**

[view plainprint?](http://www.java2blog.com/2013/04/create-restful-web-servicesjax-rs-using.html)

1. **package** org.arpit.javapostsforlearning.webservices;
2. /\*\*
3. \* @author Arpit Mandliya
4. \*/
6. **import** javax.ws.rs.GET;
7. **import** javax.ws.rs.Path;
8. **import** javax.ws.rs.PathParam;
9. **import** javax.ws.rs.Produces;
10. **import** javax.ws.rs.core.MediaType;
12. @Path("ConversionService")
13. **public** **class** FeetToInchAndInchToFeetConversionService {
14. @GET
15. @Path("/InchToFeet/{i}")
16. @Produces(MediaType.TEXT\_XML)
17. **public** String convertInchToFeet(@PathParam("i") **int** i) {
19. **int** inch=i;
20. **double** feet = 0;
21. feet =(**double**) inch/12;
23. **return** "<InchToFeetService>"
24. + "<Inch>" + inch + "</Inch>"
25. + "<Feet>" + feet + "</Feet>"
26. + "</InchToFeetService>";
27. }
29. @Path("/FeetToInch/{f}")
30. @GET
31. @Produces(MediaType.TEXT\_XML)
32. **public** String convertFeetToInch(@PathParam("f") **int** f) {
33. **int** inch=0;
34. **int** feet = f;
35. inch = 12\*feet;
37. **return** "<FeetToInchService>"
38. + "<Feet>" + feet + "</Feet>"
39. + "<Inch>" + inch + "</Inch>"
40. + "</FeetToInchService>";
41. }
42. }

**@Path(/your\_path\_at\_class\_level)** : Sets the path to base URL + /your\_path\_at\_class\_level. The base URL is based on your application name, the servlet and the URL pattern from the web.xml" configuration file.  
  
**@Path(/your\_path\_at\_method\_level)**: Sets path to base URL + /your\_path\_at\_class\_level+ /your\_path\_at\_method\_level  
  
**@Produces(MediaType.TEXT\_XML [, more-types ])**: @Produces defines which MIME type is delivered by a method annotated with @GET. In the example text ("text/XML") is produced.  
  
**@PathParam:** Used to inject values from the URL into a method parameter.This way you inject inch in convertFeetToInch method and convert that to feet.

6)Now you need to create **web.xml**and put it under **/RESTfulWebserviceExample/WebContent/WEB-INF/**

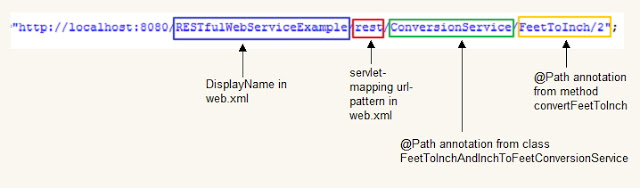
[view plainprint?](http://www.java2blog.com/2013/04/create-restful-web-servicesjax-rs-using.html)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID" version="2.5"**>**
3. **<display-name>**RESTfulWebServiceExample**</display-name>**
4. **<servlet>**
5. **<servlet-name>**Jersey REST Service**</servlet-name>**
6. **<servlet-class>**com.sun.jersey.spi.container.servlet.ServletContainer**</servlet-class>**
7. **<init-param>**
8. **<param-name>**com.sun.jersey.config.property.packages**</param-name>**
9. **<param-value>**org.arpit.javapostsforlearning.webservices**</param-value>**
10. **</init-param>**
11. **<load-on-startup>**1**</load-on-startup>**
12. **</servlet>**
13. **<servlet-mapping>**
14. **<servlet-name>**Jersey REST Service**</servlet-name>**
15. **<url-pattern>**/rest/\***</url-pattern>**
16. **</servlet-mapping>**
17. **</web-app>**

In above <param-value>,put your web service package.

8) **Run project:**right click on project->run as ->run on server

9) Test your REST service under: "http://localhost:8080/RESTfulWebServiceExample/rest/ConversionService/FeetToInch/2".

[](http://2.bp.blogspot.com/-tJ_PpvJCXvY/UV1aY0GSLwI/AAAAAAAAA9U/WEYxy3GFWkw/s1600/RestfulUrlExplained.jpg)

You will get output as :

[](http://1.bp.blogspot.com/-6eXzVf735s0/UV1be160dNI/AAAAAAAAA9g/mjm6XPTDWxU/s1600/feetToInchServiceXML.jpg)

If You see web service information page then you are done.

### Creating a Restful Web Service Client:

Create ConversionServiceClient.java under org.arpit.javapostsforlearning.websevices.client

[view plainprint?](http://www.java2blog.com/2013/04/create-restful-web-servicesjax-rs-using.html)

1. **package** org.arpit.javapostsforlearning.webservices.client;
3. **import** javax.ws.rs.core.MediaType;
4. **import** com.sun.jersey.api.client.Client;
5. **import** com.sun.jersey.api.client.ClientResponse;
6. **import** com.sun.jersey.api.client.WebResource;
7. **import** com.sun.jersey.api.client.config.ClientConfig;
8. **import** com.sun.jersey.api.client.config.DefaultClientConfig;
10. **public** **class** ConversionServiceClient {
11. **static** **final** String REST\_URI = "http://localhost:8080/RESTfulWebServiceExample";
12. **static** **final** String INCH\_TO\_FEET = "/ConversionService/InchToFeet/";
13. **static** **final** String FEET\_TO\_INCH = "/ConversionService/FeetToInch/";
15. **public** **static** **void** main(String[] args) {
17. **int** inch=12;
18. **int** feet=2;
20. ClientConfig config = **new** DefaultClientConfig();
21. Client client = Client.create(config);
22. WebResource service = client.resource(REST\_URI);
24. WebResource addService = service.path("rest").path(INCH\_TO\_FEET+inch);
25. System.out.println("INCH\_TO\_FEET Response: " + getResponse(addService));
26. System.out.println("INCH\_TO\_FEET Output as XML: " + getOutputAsXML(addService));
27. System.out.println("---------------------------------------------------");
29. WebResource subService = service.path("rest").path(FEET\_TO\_INCH+feet);
30. System.out.println("FEET\_TO\_INCH Response: " + getResponse(subService));
31. System.out.println("FEET\_TO\_INCH Output as XML: " + getOutputAsXML(subService));
32. System.out.println("---------------------------------------------------");
34. }
36. **private** **static** String getResponse(WebResource service) {
37. **return** service.accept(MediaType.TEXT\_XML).get(ClientResponse.**class**).toString();
38. }
40. **private** **static** String getOutputAsXML(WebResource service) {
41. **return** service.accept(MediaType.TEXT\_XML).get(String.**class**);
42. }
43. }

Run above program  
Output:

[view plainprint?](http://www.java2blog.com/2013/04/create-restful-web-servicesjax-rs-using.html)

1. INCH\_TO\_FEET Response: GET http://localhost:8080/RESTfulWebServiceExample/rest/ConversionService/InchToFeet/12 returned a response status of 200 OK
2. INCH\_TO\_FEET Output as XML: <InchToFeetService><Inch>12</Inch><Feet>1.0</Feet></InchToFeetService>
3. ---------------------------------------------------
4. FEET\_TO\_INCH Response: GET http://localhost:8080/RESTfulWebServiceExample/rest/ConversionService/FeetToInch/2 returned a response status of 200 OK
5. FEET\_TO\_INCH Output as XML: <FeetToInchService><Feet>2</Feet><Inch>24</Inch></FeetToInchService>

Read more at http://www.java2blog.com/2013/04/create-restful-web-servicesjax-rs-using.html#HJai0kZ47SsiGBbU.99

JAX-WS webservice interview questions

<http://www.javaexperience.com/jax-ws-webservice-interview-questions/>

# [Difference between servlet and web service](http://stackoverflow.com/questions/5930795/difference-between-servlet-and-web-service)

What you're describing is a web application, where a human uses a browser to interact with a software system.

A web service is a way for software systems to communicate with each other using HTTP and XML or JSON, without any humans involved.

A servlet is a Java-specific way of writing software that responds to HTTP requests. Spring MVC abstracts away a lot of the implementation detail to make writing web applications easier, but uses servlets under the covers.

Building a RESTFul Service with Spring MVC

<http://www.leveluplunch.com/java/tutorials/010-building-restful-webservice-spring-mvc-boot/>

A RESTful service is a way to expose your data through a URL. Let's use spring mvc to implement a RESTFul webservice to retrieve a listing of agencies.

### Detailed Video Notes

## Getting started

SOAP based web service while still serve a purpose are bloated, hard to consume directly from the client side and are bound by a hard contract. REST services on the other hand have been a go to choice for developing services consumed by web applications due to the lightweight JSON responses and soft contract. What I mean by soft contract is, REST services aren’t bound to a WSDL but still have a contract in which you need to abide by or have a versioning strategy in place. Using [**spring boot**](http://projects.spring.io/spring-boot/) as a foundation lets build a simple REST web service.

## What we will build

[[**0:37**](http://www.youtube.com/embed/bdTsY_7SEXw?start=37&autoplay=1)]

While not overly exciting lots of folks are familiar with insurance, agencies sell policies to policyholders. Lets say we want to get all the possible agencies we can purchase a policy from. We will make a request tohttp://localhost:8080/agencies which should return the following:

[

{

"id":1,

"name":"All State",

"ein":"123"

},

{

"id":2,

"name":"FCCI Insurance Group",

"ein":"456"

},

{

"id":3,

"name":"Farmers",

"ein":"789"

},

{

"id":4,

"name":"Met life",

"ein":"167"

}

]

## Project set up

[[**0:54**](http://www.youtube.com/embed/bdTsY_7SEXw?start=54&autoplay=1)]

We will use [**spring STS**](http://spring.io/tools) to create project from a new spring starter project. We will select web project which will include the necessary starter projects in our pom.xml

## Create resource representation class

[[**1:4**](http://www.youtube.com/embed/bdTsY_7SEXw?start=64&autoplay=1)]

Depending on your philosophy of building services, you could return domain object from your business logic layer or create a class that represents the resource a client is requesting. In our case lets create an agency resource object that contains a small subset of an agency object.

public class AgencyResource {

private Integer id;

private String name;

private String EIN;

public AgencyResource(Integer id, String name, String eIN) {

super();

this.id = id;

this.name = name;

EIN = eIN;

}

//...

}

## Create resource controller

[[**1:21**](http://www.youtube.com/embed/bdTsY_7SEXw?start=81&autoplay=1)]

Next, we need something to handle the request and direct it to a method for processing. Within spring, HTTP requests are handled by a controller. These components are identified by the @Controller annotation. The@RequestMapping annotation ensures that HTTP requests to /agencies are mapped to the getAgencies() method.

A key difference between a traditional MVC controller and the RESTful web service controller is the way that the HTTP response is created. Rather than delegating to the view layer we want the object data written directly to the HTTP response as JSON. To accomplish this, the @ResponseBody annotation on the getAgencies() method tells Spring MVC that it does not need to look for a view and the list of agencies can be returned directly to the response.

Before the java object can be sent to the response it needs to be converted to JSON. This is done by an[**HttpMessageConverter**](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/http/converter/HttpMessageConverter.html) specifically Jackson’s [**MappingJackson2HttpMessageConverter**](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/http/converter/json/MappingJackson2HttpMessageConverter.html) that it picks up on the class path. If you are not familiar with jackson, jackson is a java library to marshal and unmarshal json.

@Controller

public class AgencyController {

@RequestMapping("/agencies")

@ResponseBody

public List<AgencyResource> getAgencies() {

List<AgencyResource> agencies = getListing();

return agencies;

}

//...

}

Spring 4 introduced a @RestController which is a @Controller and a @ResponseBody in one which eliminates the need to specify @ResponseBody on each method. Notice below that the @Controller has been changed to @RestController and the @ResponseBody has been removed.

@RestController

public class AgencyController {

@RequestMapping("/agencies")

public List<AgencyResource> getAgencies() {

List<AgencyResource> agencies = getListing();

return agencies;

}

//..

}

## Creating mock data

Before we get to running the application, we will create some sample data that we can send in the response.

//...

private List<AgencyResource> getListing() {

List<AgencyResource> resources = new ArrayList<>();

resources.add(new AgencyResource(1, "All State", "123"));

resources.add(new AgencyResource(2, "FCCI Insurance Group", "456"));

resources.add(new AgencyResource(3, "Farmers", "789"));

resources.add(new AgencyResource(4, "Met life", "167"));

return resources;

}

//...

## Running the application

[[**2:51**](http://www.youtube.com/embed/bdTsY_7SEXw?start=171&autoplay=1)]

Running the application the default starter project was nice enough to create an Application.java class. Lets break down what is happening in spring.

The main() method defers to the SpringApplication helper class, providing Application.class as an argument to its run() method. This tells Spring to read the annotation metadata from the Application and to manage it as a component in the Spring application context. In other words it is used to bootstrap and launch the Spring application from a java main method. This might look a bit different than the traditional way to deploy to an application server.

The @ComponentScan annotation is used to look and locate other spring beans within the demo package. This directive ensures that Spring finds and registers the AgencyController as it is marked with a with @Controllerannotation which is a type of @Component.

The @EnableAutoConfiguration annotation is class that enables spring boots convention over configuration which enables default behaviors by scanning the classpath. For example, as we noted above a java object will be marshalled to json by jackson. This is because spring boot found jackson in the class path. In addition, since the project depends on Spring MVC (spring-webmvc.jar), a Spring MVC DispatcherServlet is configured and registered for you through the DispatcherServletAutoConfiguration class.

Right clicking on the project and running the class will start the application in a tomcat container. Navigating to localhost:8080/agencies we will perform a GET and the list of agencies is returned.

Thanks for joining in today’s level up lunch, have a great day!

[JAX-RS](http://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services) is the Java api for restful webservice. [Jersey](http://jersey.java.net/) is an implementation from sun/oracle.

**[calling Restful Service from Java](http://stackoverflow.com/questions/7177628/calling-restful-service-from-java)**

<http://stackoverflow.com/questions/7177628/calling-restful-service-from-java>

Well, there are many ways you could access RESTful API using Java third party libraries, like using [Apache Http Client](http://hc.apache.org/httpcomponents-client-ga/index.html) or using [Sun Jersey API](http://blogs.oracle.com/enterprisetechtips/entry/consuming_restful_web_services_with).

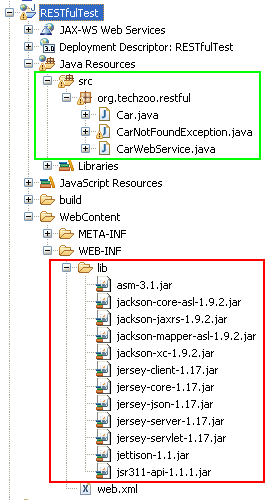
You could also use the native java.net.HttpURLConnection to consume REST Services

**RESTful java web service with XML Response**

<http://www.techzoo.org/java/java-web-services/restful-java-web-service-with-xml-response.html>

In my previous post I discuss about how to write a [simple REST web service using jersey](http://www.techzoo.org/how-to/how-to-create-simple-rest-web-service-using-jersey.html), today we discuss more on it and demonstrate how you can create custom java class with JAXB and deal with XML response using jersey.

Let’s first create eclipse Java EE project and add all depending jersey jar files in classpath.



Edit your web.xml to look like similar to following.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"      xmlns="http://java.sun.com/xml/ns/javaee"xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"      id="WebApp\_ID" version="3.0">      <display-name>RESTfulTest</display-name>        <servlet>          <servlet-name>jersey-serlvet</servlet-name>          <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>          <init-param>              <param-name>com.sun.jersey.config.property.packages</param-name>              <param-value>org.techzoo.restful</param-value>          </init-param>          <load-on-startup>1</load-on-startup>      </servlet>        <servlet-mapping>          <servlet-name>jersey-serlvet</servlet-name>          <url-pattern>/rest/\*</url-pattern>      </servlet-mapping>    </web-app> |

We create Car class and make it JAXB compatible so it will be easy for jersey to return response in XML-format. Your class should like below. The annotation is self-explanatory.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | package org.techzoo.restful;    import javax.xml.bind.annotation.XmlAttribute;  import javax.xml.bind.annotation.XmlElement;  import javax.xml.bind.annotation.XmlRootElement;    @XmlRootElement(name="car-details")  public class Car {        private int carId;      private String carName, make;      private double price;        public Car(){}        public Car(int carId, String carName,          String make, double price) {          super();          this.carId = carId;          this.carName = carName;          this.make = make;          this.price = price;      }        @XmlAttribute(name = "car-id")      public int getCarId() {return carId;}      public void setCarId(int carId)      {this.carId = carId;}        @XmlElement (name="car-name")      public String getCarName() {return carName;}      public void setCarName(String carName)      {this.carName = carName;}        @XmlElement      public String getMake() {return make;}      public void setMake(String make)      {this.make = make;}        @XmlElement      public double getPrice() {return price;}      public void setPrice(double price)      {this.price = price;}        @Override      public String toString() {          return String.format("Car Details " +              "[Name=%s, Make=%s, Price=%f]",              carName, make, price);      }    } |

Customized exception class for proper exception propagation to client.



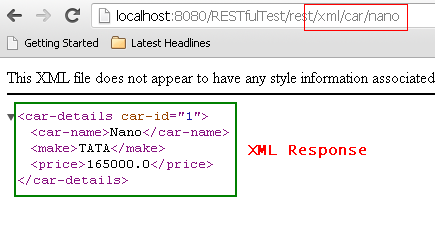
|  |  |
| --- | --- |
| 1  2  3  4  5  6 | package org.techzoo.restful;    public class CarNotFoundException extends RuntimeException {      public CarNotFoundException(){super();}      public CarNotFoundException(String msg){super(msg);}  } |

Now write web service class which uses Car class and return response to client in XML format. We use@Produces (MediaType.APPLICATION\_XML) annotation for returning XML format.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | package org.techzoo.restful;    import java.util.HashMap;  import java.util.Map;    import javax.ws.rs.GET;  import javax.ws.rs.Path;  import javax.ws.rs.PathParam;  import javax.ws.rs.Produces;  import javax.ws.rs.core.MediaType;    @Path("/xml/car")  public class CarWebService {        private Map<String, Car> cars;        public CarWebService(){          cars = new HashMap<String, Car>();          cars.put("nano", new Car(1,"Nano","TATA",165000));          cars.put("tavera", new Car(2, "Tavera", "Chervolet", 865000));          cars.put("scorpio", new Car(2, "Scorpio", "Mahindra", 870000));      }        @GET      @Path("/{name}")      @Produces (MediaType.APPLICATION\_XML)      public Car getCarDetailsByName(@PathParam("name") String name) {          if(cars.containsKey(name)) return cars.get(name);          else throw new CarNotFoundException(              "Car with name = "+name+" not found.");      }        @GET      @Path("/{name}/{make}")      @Produces (MediaType.APPLICATION\_XML)      public Car getCarDetailsByNameAndMake(@PathParam("name") String name,              @PathParam("make") String make)      {          if(cars.containsKey(name)){              Car car = cars.get(name);              if(car.getMake().equals(make)) {                  return car;              }              else throw new CarNotFoundException(                      "Car with name = "+name+" doesn't belong to "+make);          }          else throw new CarNotFoundException(              "Car with name = "+name+" not found.");      }    } |

After publishing this web service you can access this through browser using following URL.  
http://localhost:8080/RESTfulTest/rest/xml/car/nano



Write java client to access this web service…



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | package org.techzoo.clients;    import org.techzoo.restful.Car;    import com.sun.jersey.api.client.Client;  import com.sun.jersey.api.client.ClientResponse;  import com.sun.jersey.api.client.WebResource;    /\*\*  \* @author Tousif Khan  \*\*/  public class CarWebServiceClient {        private static final String URL =          "http://localhost:8080/RESTfulTest/rest/xml/car/scorpio";        public static void main(String[] args) throws Exception {          try {              Client client = Client.create();              WebResource webResource = client.resource(URL);              WebResource.Builder builder = webResource.accept("application/xml");              ClientResponse response = builder.get(ClientResponse.class);              if (response.getStatus() != 200) {                  throw new RuntimeException("Failed : HTTP error code : "+          response.getStatus());              }              Car car = response.getEntity(Car.class);              System.out.println(car);          } catch (Exception e) {              e.printStackTrace();          }      }    } |

LOOK AT THE FOLLING SITES FOR EXAMPLES:

<http://www.vogella.com/tutorials/REST/article.html>

<http://stackoverflow.com/questions/20784531/jersey-services-with-tomcat-and-eclipse>

JERSEY LIBRARY:

<http://repo1.maven.org/maven2/org/glassfish/jersey/bundles/jaxrs-ri/2.5/jaxrs-ri-2.5.zip>

# [Difference between RESTful webservice and HttpServlet](http://stackoverflow.com/questions/14314658/difference-between-restful-webservice-and-httpservlet)

REST is really an architectural style used when designing an API on a server. HttpServlets can be a method of implementing a RESTful web service.

REST describes a style where HTTP verbs like GET/POST/DELETE/etc. are used in a predictable way to interact with resources on a server.

I'd recommend reading through the [REST Wikipedia](http://en.wikipedia.org/wiki/REST) page for a good overview.

### Increase quality and productivity with the Jersey Test Framework

<http://usna86-techbits.blogspot.com/2013/03/increase-quality-and-productivity-with.html>

## Friday, March 1, 2013

### Increase quality and productivity with the Jersey Test Framework

*Note:  This article applies to Jersey 1.x.  If you're looking for information on how to use the Jersey Test Framework in Jersey 2 please see*[*this more recent article.*](http://usna86-techbits.blogspot.com/2013/12/jersey-jerseytest-migration-from-1x-to.html)  
  
With the Jersey test framework developers can increase the quality of their software as well as their productivity without leaving the comfort of their favorite IDE.  
  
The framework spins up an embedded servlet container that is configured to load the restful resources specified by the developer. In addition, the SpringServlet can be used to wire in the necessary beans if Spring is being used.  
  
And, this is really super simple. The key is to extend the JerseyTest class and override the configure() method. In the configure() method you supply the same information that you would normally provide in your web.xml.   
  
Line 5 : specify the package that contains the Jersey resource(s) you want to test  
Line 6 : provide the name and location of the Spring context file (if using Spring)  
Line 7 : turn on the JSON to POJO mapping feature if you want to use that

[?](http://usna86-techbits.blogspot.com/2013/03/increase-quality-and-productivity-with.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public class MyResourceWebServiceTest extends JerseyTest {        @Override      protected AppDescriptor configure() {          return new WebAppDescriptor.Builder("com.mycompany.services")              .contextParam("contextConfigLocation", "classpath:\*\*/testContext.xml")              .initParam("com.sun.jersey.api.json.POJOMappingFeature", "true")              .servletClass(SpringServlet.class)              .contextListenerClass(ContextLoaderListener.class)              .requestListenerClass(RequestContextListener.class)              .build();      } // configure() |

Once your test class is configured to spin up the embedded web container with your resources now it's time to write your tests. Again, the Jersey test framework makes it so easy even a caveman can do it.  
  
On line 3 below we simply access a WebResource object and provide the relative URI to the resource we are interested in. This URI should match the @Path mappings in your Jersey resource definition.  
  
Once the WebResource is defined simply use it to build and execute the HTTP request for the desired HTTP method, in this case GET, as shown on line 6. That's it. All that's left to do is the standard JUnit stuff to validate the response.

[?](http://usna86-techbits.blogspot.com/2013/03/increase-quality-and-productivity-with.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | @Test  public void someTest() {      WebResource webResource = resource().path("/some/resource/17");      ClientResponse response =  webResource          .accept(MediaType.APPLICATION\_JSON)          .get(ClientResponse.class);        assertEquals(200, response.getStatus());        try {          JSONObject obj = new JSONObject(response.getEntity(String.class));          assertEquals("widget", obj.get("type"));      } catch (JSONException e) {          fail(e.getMessage());      }  } // someTest() |

Now, push a button or hit a key or two to kick off your JUnit test suite and watch your Jersey web services and tests fly.  If you need to make a change it only takes a minute or two to modify your code and run the tests again.  
  
One piece of advice - create a test specific Spring context file targeting the exact REST resources you want to test and, if your resources eventually end up accessing some datasource (most would), consider injecting mock data access objects into your Spring beans so you can easily control the data that your resource would have access to, thus easily facilitating your testing (and development) and making your tests repeatable.  
  
See [this post](http://usna86-techbits.blogspot.com/2013/02/how-to-return-location-header-from.html) if you want to learn how to easily create hyperlinks in your Jersey REST services.  
  
See [this post](http://usna86-techbits.blogspot.com/2013/08/restful-java-web-service-marshalling.html) if you want to see more examples of Jersey unit testing or comparisons of different ways to marshall your data/objects.

# [What's the difference between a POST and a PUT HTTP REQUEST?](http://stackoverflow.com/questions/107390/whats-the-difference-between-a-post-and-a-put-http-request)

1) GET:- Used when the client is requesting a resource on the Web server.

2) HEAD:- Used when the client is requesting some information about a resource but not requesting the resource itself.

3) POST:- Used when the client is sending information or data to the server—for example, filling out an online form (i.e. Sends a large amount of complex data to the Web Server).

4) PUT:- Used when the client is sending a replacement document or uploading a new document to the Web server under the request URL.

5) DELETE:- Used when the client is trying to delete a document from the Web server, identified by the request URL.

6) TRACE:- Used when the client is asking the available proxies or intermediate servers changing the request to announce themselves.

7) OPTIONS:- Used when the client wants to determine other available methods to retrieve or process a document on the Web server.

8) CONNECT:- Used when the client wants to establish a transparent connection to a remote host, usually to facilitate SSL-encrypted communication (HTTPS) through an HTTP proxy.

Securing RESTful Web Services

<http://docs.oracle.com/cd/E24329_01/web.1211/e24983/secure.htm#RESTF113>

* [About RESTful Web Service Security](http://docs.oracle.com/cd/E24329_01/web.1211/e24983/secure.htm#BABCBAJJ)
* [Securing RESTful Web Services Using web.xml](http://docs.oracle.com/cd/E24329_01/web.1211/e24983/secure.htm#BABDGCJD)
* [Securing RESTful Web Services Using SecurityContext](http://docs.oracle.com/cd/E24329_01/web.1211/e24983/secure.htm#BABHABCC)
* [Securing RESTful Web Services Using Annotations](http://docs.oracle.com/cd/E24329_01/web.1211/e24983/secure.htm#BABGBJAC)

# [How to implement a HTTPS login page in a web application?](http://stackoverflow.com/questions/1454021/how-to-implement-a-https-login-page-in-a-web-application)

<http://stackoverflow.com/questions/1454021/how-to-implement-a-https-login-page-in-a-web-application>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11down vote[favorite](http://stackoverflow.com/questions/1454021/how-to-implement-a-https-login-page-in-a-web-application)  **18** | I want to create a secure login/logout mechanism. I started reading the following articles to get an idea of things to take into account:   * [Solving the Logout Problem Properly and Elegantly](http://www.javaworld.com/javaworld/jw-09-2004/jw-0927-logout.html) * [Revisiting the logout problem](http://www.javaworld.com/javaworld/jw-10-2006/jw-1006-logout.html)   These articles make some good points, but I was thinking in using HTTPS in a similar way as the Yahoo mail login page. You know... you type [http://mail.yahoo.com](http://mail.yahoo.com/) and you are redirected to a HTTPS page like**https://**login.yahoo.com/config/login where you insert your username and password and after your credentials are verified you are redirected back to a HTTP page with a generated session\_id cookie and all communications from there on are on HTTP using the cookie.  What do I need to implement this behavior?  I want to do this for two Java web apps (one with Spring framework and one with Struts 1) but don’t know exactly how to integrate that HTTPS part into the application (I have never worked with HTTPS before).  [java](http://stackoverflow.com/questions/tagged/java) [login](http://stackoverflow.com/questions/tagged/login) [https](http://stackoverflow.com/questions/tagged/https)   |  |  |  | | --- | --- | --- | | [share](http://stackoverflow.com/q/1454021/3426143)|[edit](http://stackoverflow.com/posts/1454021/edit) | [edited Sep 24 '09 at 8:18](http://stackoverflow.com/posts/1454021/revisions) | asked Sep 21 '09 at 11:14  user159088 | |
|  | add a comment |

## 6 Answers

[active](http://stackoverflow.com/questions/1454021/how-to-implement-a-https-login-page-in-a-web-application?answertab=active#tab-top)[oldest](http://stackoverflow.com/questions/1454021/how-to-implement-a-https-login-page-in-a-web-application?answertab=oldest#tab-top)[votes](http://stackoverflow.com/questions/1454021/how-to-implement-a-https-login-page-in-a-web-application?answertab=votes#tab-top)

|  |  |
| --- | --- |
| up vote15down voteaccepted  +100 | First of all you need to enable SSL for your server. For Tomcat you need to generate an open SSL keystore and add the following connector to server.xml:  <Connector port="8443" scheme="https" secure="true" SSLEnabled="true"  keystoreFile="mykeystore" sslProtocol="TLS"  keystorePass="keystore password" />  To integrate SSL into your application I recommend Spring Security. It offers exactly what you want (login over HTTPS, then redirected to HTTP). All you have to do to implement it, is to set force HTTPS to true:  <bean id="authenticationProcessingFilterEntryPoint"  class="org.springframework.security.ui.webapp.AuthenticationProcessingFilterEntryPoint">  <property name="loginFormUrl" value="/pages/login.jsp" />  <property name="forceHttps" value="true"/>  </bean>  Of course Spring and Spring security do have a rather steep learning curve, but it is totally worth it. Do it once and then you can apply it to new apps in less than an hour. You can use Spring Security in both the Spring and Struts application.  Spring security used to be Acegi security. This is an [article](http://www.javaworld.com/javaworld/jw-10-2007/jw-10-acegi2.html?page=1) that will get you started. |

Not sure about any Java or spring specifics, but in general:

1) Set up an SSL cert on your server.

2) Forward or Link to an absolute URL (with https:// at the beginning) when going to login page

3) Forward to an absolute URL (with http://) after successful authentication.

4) Include a check in the login page code to only accept https connections.

Of course there may be framework specific ways of doing the http/https redirect without resorting to explicitly specifying the full URL.