**What is Web Services, Web Services Introduction**

What is Web Services ? Over the internet, you might have seen different kinds of definitions for Web services. My definition will almost resembles them :-)Web Services, the name it self indicates that its a service which is available over the Web, that’s it. As an example you can consider Java4s.com, When ever you hit the URL in the web browser it will gives you some output in HTML format, you can also consider this as a Web service.  With web services, we can communicate different applications on different platforms, i mean a java application in Windows platform can easily communicate with the application developed using .net/php in Linux operation system.

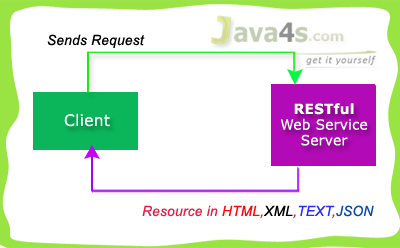
## Understanding SOAP and REST

Web Services are mainly of 2 types, **SOAP** [Simple Object Access Protocol] and **REST** [Representational state transfer] based services. We have different type of specifications to implement SOAP and REST services. I believe so far you might be in confusion with these kind keywords like, JAX-RS, JAX-WS, RESTful, SOAP, Apache Axis2, Apache CXF bla bla…  Let me try to bring you out of them.

* JAX-RS provides the implementation of RESTful web services,  JAX-RS is a specification for RESTful Web Services with Java and it is given by Sun.  Since it is a specification, other frameworks can be written to implement these specifications, and that includes Jersey from Oracle, Resteasy from Jboss, CXF from Apache bla bla.
* JAX-WS, Apache Axis2 provides the implementation for SOAP
* Apache CXF provides implementation for SOAP and RESTful services both.

## RESTful

What ever the data/response we will get from the server is known as ***Resource*** [remember this point], Each resource can be accessed by its URI’s.  We can get the resource from RESTful service in different formats like, HTML,XML,JSON,TEXT,PDF and in the Image formats as well, but in real time we mainly we will prefer JSON.  REST guidelines always talks about stateless communication between client and the Server.  Stateless means, every single request from client to server will be considered as a fresh request. Because of this reason REST always prefers to choose HTTP as it a stateless protocol.



RESTful used 4 main HTTP methods…

* **GET** - Retrieve Data
* **POST**- Create/Insert Data
* **PUT**- Update Data
* **DELETE**- Delete Data

Generally we will prefer RESTful Services in these scenarios…

* If clients require caching, means if you have limited bandwidth
* If you want every thing to be stateless [ I have already explained about stateless ]

But SOAP gives the output only in XML format.   Hope you are good now :-)by the way we are going to use Jersey to implement JAX-RS specifications.

# RESTful Web Services (JAX-RS) Annotations

This tutorial explains important annotations of JAX-RS for creating RESTful web services, friends i am giving these annotations just for your understanding purpose. you better know about these annotations before we go forward with the remaining RESTful web services tutorials.

## JAX-RS Annotations

* @Path(‘Path‘)
* @GET
* @POST
* @PUT
* @DELETE
* @Produces(MediaType.TEXT\_PLAIN [, more-types])
* @Consumes(type[, more-types])
* @PathParam()
* @QueryParam()
* @MatrixParam()
* @FormParam()

## @Path() Annotation

* Its a Class & Method level of annotation
* This will check the path next to the base URL

**Syntax** :  
Base URL :  
http://localhost:(port)/<YourApplicationName>/<UrlPattern In Web.xml>/<path>  
Here <path> is the part of URI, and this will be identified by @path annotation at class/method level, you will be able to understand in the next RESTful  hello world tutorial.

## @GET

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP GET request only,  i mean if we annotate our method with @GET, the execution flow will enter that following method if we send GET request from the client

## @POST

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP POST request only.

## @PUT

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP PUT request only.

## @DELETE

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP DELETE request only.

## @Produces

Its a method or field level annotation, This tells which MIME type is delivered by the method annotated with @GET.  I mean when ever we send a HTTP GET request to our RESTful service, it will invokes particular method and produces the output in different formats.  There you can specifies in what are all formats (MIME) your method can produce the output, by using @produces annotation.  
**Remember**: We will use @Produces annotation for GET requests only.

## @Consumes

This is a class and method level annotation, this will define which MIME type is consumed by the particular method. I mean in which format the method can accept the input from the client.

Will discuss later regarding @PathParam, @QueryParam, @MatrixParam, @FormParam annotations :-), i will talk more about these annotations in the next examples.

# How RESTful Web Services Extract Input Parameters

In this article i will show you how a RESTful web service will  extract input parameters from the client request.  We have different ways of sending input values to the rest services, and RESTful web service extract those details based upon the client URL pattern. In JAX-RS we can use the following annotations to extract the input values sent by the client.

* @PathParam
* @QueryParam
* @MatrixParam
* @FormParam

@PathParam,@QueryParam,@MatrixParam are parameter annotations which allows us to map variable URI path fragments into your method call. Confused ? :-)In simple words, these three annotations will come into picture in case if we are passing the input values to the restful service through the URL. After that Rest service will extract those values by using these annotations. Regarding @FormParam, restful web service will use this annotation to retrieve the values sent by the client through some HTML/JSP form.

## @PathParam URL Syntax

# http://localhost:7001/<Rest Service Name>/rest/customers/100/Java4s

Did you observe the two parameters appear in the end of the above URL [100 & Java4s], which are separated by forward slash(/) are called as path parameters, as of now just remember the syntax, going forward i will give you an example on each annotation.

## @QueryParam URL Syntax

# http://localhost:7001/…/rest/customers?custNo=100&custName=Java4s

If the client sends an input in the form of query string in the URL, then those parameters are called as Query Parameters.  If you observe the above syntax, client passing 2 parameters 100 and Java4s  started after question mark (?) symbol and each parameter is separated by & symbol,  those parameters are called as query parameters.

## @MatrixParam URL Syntax

# http://localhost:7001/…/rest/customers;custNo=100;custName=Java4s

Matrix parameters are  another way defining the parameters to be added to URL.  If you observe the above syntax, client is passing two parameters each are separated by semicolon, these parameters are called as matrix parameters.  Remember these parameters may appear any where in the path.

## @FormParam URL Syntax

Finally form parameters,  if we have a HTML form having two input fields and submit button. Lets client enter those details and submit to the RESTful web service. Then the rest service will extract those details by using this @FormParam annotation.  
For now just remember these consents, going forward i will give you an example on each annotation.

# Jersey Hello World Example Using JAX-RS Specification

## HelloWorldService.java

package com.java4s;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.Response;

@Path("/customers")

public class HelloWorldService {

  @GET

  @Produces("text/html")

  public Response getLocalCust() {

           String output = "I am from 'getLocalCust' method";

           return Response.status(200).entity(output).build();

  }

  @GET

  @Path("/nri")

  @Produces("text/html")

  public Response getNriCust() {

            String output = "I am from 'getNriCust' method";

            return Response.status(200).entity(output).build();

  }

}

## web.xml

<web-app id="WebApp\_ID" version="2.4"

xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<display-name>RestPathAnnotationExample</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>com.java4s</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>

Now we are good to run the application, Just right click on the project >Run As > Run on Server > It will open the application URL like  
***http://localhost:2013/RestPathAnnotationExample/***

But you need to satisfy the actual URL pattern, i mean change the URL to…

Main application URL:  
http://localhost:2013/RestPathAnnotationExample/

Web.xml URL pattern:  
/rest

@Path in HelloWorldService.java:  
/customers

Final URL should be  
**http://localhost:2013/RestPathAnnotationExample/rest/customers**

# RESTful Web Services (JAX-RS) @PathParam Example

package com.java4s;

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.Response;

@Path("/customers")

public class RestServicePathParamJava4s {

    @GET

    @Path("{name}/{country}")

    @Produces("text/html")

    public Response getResultByPassingValue(

                    @PathParam("name") String name,

                    @PathParam("country") String country) {

        String output = "Customer name - "+name+", Country - "+country+"";

        return Response.status(200).entity(output).build();

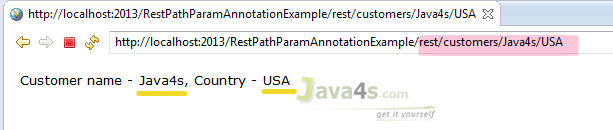
    }

}

## Explanation

* Right click on your project > Run As > Run on Server
* By default eclipse will open http://localhost:2013/RestPathParamAnnotationExample/ with  HTTP 404 Error
* In web.xml we have specified URL pattern as /rest/\* (line number 19) and in RestServicePathParamJava4s.java we specified class level @path as /customers [ line number 9 ] and method level @path as  {name}/{country} [ line number 13 ]
* So the final URL should be http://localhost:2013/RestPathParamAnnotationExample/rest/customers/Java4s/USA
* Once you hit the URL,  http://localhost:2013/……/rest/../Java4s/USA  , last two parameters in this URL ‘Java4s‘ and ‘USA‘ are retrieved by @PathParam(“name”), @PathParam(“country”) annotations in RestServicePathParamJava4s.java and will copy into String name, String country respectively.  
  
* Check the output

## Output



# RESTful Web Services (JAX-RS) @QueryParam Example

* In RESTful web services (JAX-RS) @QueryParam annotation will be used to get the query parameters from the URL, Observe carefully, i am saying we will retrieve the parameters only not their values.  But in case of @PathParam we will get parameter values directly.

## Query Parameters Syntax

Consider this URL:  
http://localhost:2013/RestPathAnnotationExample/rest/customers?nameKey=Java4s&countryKey=USA  
Here query parameters are nameKey, countryKey and their values are Java4s, USA respectively, hope you understood.

## Required Files

* pom.xml & web.xml are similar to [previous article](http://www.java4s.com/hibernate/restful-web-services-jax-rs-pathparam-example/) no changes :-)
* RestServiceQueryParamJava4s.java

## RestServiceQueryParamJava4s.java

@Path("/customers")

public class RestServiceQueryParamJava4s {

    @GET

    @Produces("text/html")

    public Response getResultByPassingValue(

                    @QueryParam("nameKey") String name,

                    @QueryParam("countryKey") String country) {

        String output = "Customer name - "+name+", Country - "+country+"";

        return Response.status(200).entity(output).build();

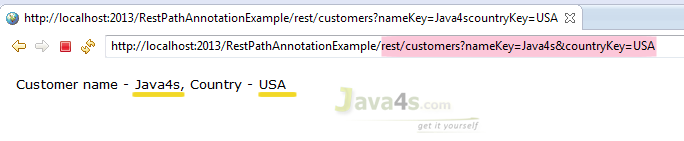
    }

}

## Explanation

* Right click on your project folder > Run As > Run on Server
* Eclipse will open http://localhost:2013/RestQueryParamAnnotationExample/ with  HTTP 404 Error
* In web.xml we have specified the url pattern as /rest/\* (line number 19) and in RestServiceQueryParamJava4s.java we specified class level @path as /customers [ line number 9 ] and we are retrieving 2 query parameters [ Line number 15,16 ], so our final URL should be the http://localhost:2013/RestQueryParamAnnotationExample/rest/customers?nameKey=java4s&countryKey=USA
* Check the output by hitting the above URL\

## Output



# RESTful Web Services (JAX-RS) @MatrixParam Example

* In this article i will describe how a RESTful web services would accept multiple parameters sent by the client in the HTTP URL as Matrix Params. So what are matrix parameters ? let me give you the syntax

## Matrix Parameters Syntax

Consider this URL  
http://localhost:2013/<projectRoot>/rest/customers;nameKey=Java4s;countryKey=USA

If you observe the URL, i am passing 2 parameters nameKey=Java4s & countryKey=USA.  One parameter is separated from another with a semicolon, similarly you can pass any number of parameters. These type of parameters are called as Matrix Parameters. I will explain more about matrix parameters in this example.

## Required Files

* web.xml & pom.xml [same as [previous articles](http://www.java4s.com/web-services/restful-web-services-jax-rs-pathparam-example/)]
* RestServiceMatrixParamJava4s.java

## RestServiceMatrixParamJava4s.java

@Path("/customers")

public class RestServiceMatrixParamJava4s{

    @GET

    @Produces("text/html")

    public Response getResultByPassingValue(

                    @MatrixParam("nameKey") String name,

                    @MatrixParam("countryKey") String country) {

        String output = "Customer name - "+name+", Country - "+country+"";

        return Response.status(200).entity(output).build();

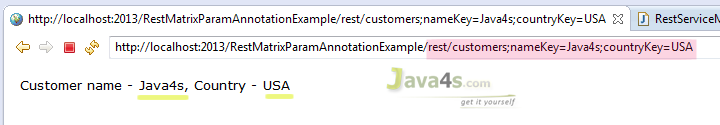
    }

}

## Explanation

* Once you run the application,  eclipse will open the following URL  http://localhost:2013/RestMatrixParamAnnotationExample/   by default
* In RestServiceMatrixParamJava4s.java [ line number 9 ] we have given class level path as /customers and we are using @MatrixParam annotation to retrieve the client inputs from the URL, so the final URL should be  
  http://localhost:2013/<projectRoot>/rest/customers;nameKey=Java4s;countryKey=USA

## Output



# RESTful Web Services (JAX-RS) @FormParam Example

By using @FormParam annotation, RESTful web service would accept HTML form parameters sent by the client in the POST request and bind them to the method variables. Generally @FormParam will come into picture when client send the data in POST request, if its the GET request @QueryParam would be the best choice.

## RestServiceFormParamJava4s.java

@Path("/customers")

public class RestServiceFormParamJava4s {

    @POST

    @Path("/addCustomer")

    @Produces("text/html")

    public Response getResultByPassingValue(

                    @FormParam("nameKey") String name,

                    @FormParam("countryKey") String country) {

        String output = "<font face='verdana' size='2'>" +

                "Web Service has added your Customer information with Name - <u>"+name+"</u>, Country - <u>"+country+"</u></font>";

        return Response.status(200).entity(output).build();

    }

}

## Client.html

<html>

<head>

    <title>RESTful Web Services (JAX-RS) @FormParam Exampale</title>

</head>

<body>

    <form action="http://localhost:2013/RestFormParamAnnotationExample/rest/customers/addCustomer" method="post">

        <table>

            <tr>

                <td><font face="verdana" size="2px">Customer Name : </font></td>

                <td><input type="text" name="nameKey" /> </td>

            </tr>

            <tr>

                <td><font face="verdana" size="2px">Country</font></td>

                <td> <input type="text" name="countryKey" /> </td>

            </tr>

            <tr>

                <td></td>

                <td><input type="submit" value="Add Customer" /> </td>

            </tr>

        </table>

    </form>

</body>

</html>

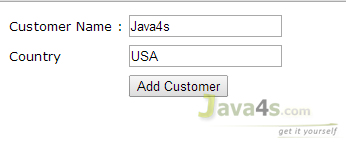
## Explanation

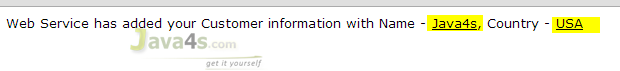
* Right click on your project root folder > Run As > Run on Server
* Eclipse will open http://localhost:2013/<projectRootFolder> with 404 Error by default, forget about that
* Now open Client.html in your web browser, enter the details and click submit [ I have created this .html file to send input form parameters to our RESTful service, you no need to create & place this file in the project workspace, myself i have created client.html file in my desktop and open in Google chrome, and verified the output]
* In Client.html, observe the URL in the from action [ line number **7** ]
* Once you click on Submit, Client.html will POST the data to the restful service. From there REST service will retrieve those details by using @FormParam annotation.

**Remember**:  Input field names in Client.html [ line numbers 12,17 ] should match with @FormParam(“-“) parameters[ line numbers 16,17 ] in RestServiceFormParamJava4s.java

## Output

Input:



Output:  


# JAX-RS Example of Multiple Resource Formats

This article will describe how a RESTful web service produces multiple output formats.  In the previous articles we came across how a RESTful service produces either XML or JSON alone as an output, but in this article i will show you what are the changes or steps need to be followed to let the rest service to produce multiple output formats from a ***single*** method, its the real time way of creating the services.  Before you start reading this article, just have a look at the following articles for better understanding.

## web.xml

<?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/j2ee" xmlns:javaee="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/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd" id="WebApp\_ID" version="2.4">

  <display-name>JAX-RS-Multiple-Output-Formats</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>java4s</param-value>

        </init-param>

        <init-param>

          <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>

          <param-value>true</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>

## Customer.java

package java4s;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

import com.sun.xml.txw2.annotation.XmlAttribute;

@XmlRootElement(name = "customer")

public class Customer {

    private int custNo;

    private String custName;

    private String custCountry;

    @XmlAttribute

    public int getCustNo() {

        return custNo;

    }

    public void setCustNo(int custNo) {

        this.custNo = custNo;

    }

    @XmlElement

    public String getCustName() {

        return custName;

    }

    public void setCustName(String custName) {

        this.custName = custName;

    }

    @XmlElement

    public String getCustCountry() {

        return custCountry;

    }

    public void setCustCountry(String custCountry) {

        this.custCountry = custCountry;

    }

}

## JsonFromRestful.java

package java4s;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.Produces;

@Path("/customers")

public class JsonFromRestful {

    @GET

    @Path("/{cusNo}")

    @Produces("application/xml,application/json")

    //@Produces(MediaType.APPLICATION\_JSON)

    public Customer produceCustomerDetailsinJSON(@PathParam("cusNo") int no) {

        Customer cust = new Customer();

            cust .setCustNo(no);

            cust .setCustName("Java4s");

            cust .setCustCountry("India");

        return cust;

    }

}

## RESTfulClient.java

package java4s;

import com.sun.jersey.api.client.Client;

import com.sun.jersey.api.client.ClientResponse;

import com.sun.jersey.api.client.WebResource;

public class RESTfulClient {

    public static void main(String[] Java4s) {

        try {

            Client client = Client.create();

            WebResource resource = client.resource("http://localhost:2015/JAX-RS-Multiple-Output-Formats/rest/customers/100");

            ClientResponse response = resource.accept("application/json").get(ClientResponse.class);

            if(response.getStatus() == 200){

                String output = response.getEntity(String.class);

                System.out.println(output);

            }else System.out.println("Somthing went wrong..!");

          } catch (Exception e) {

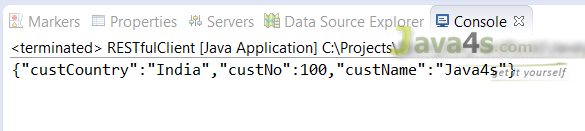
                  e.printStackTrace();

          }

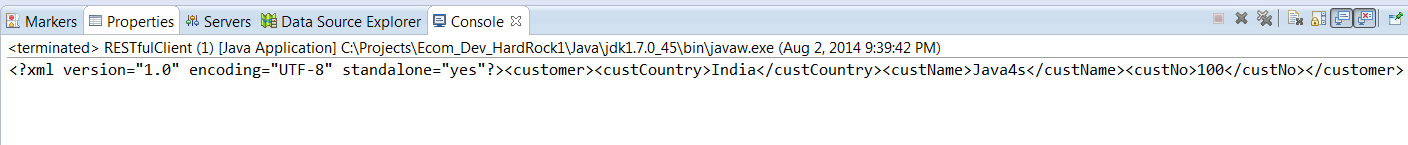
        }

}

## Execution & Output

* Eclipse > right click on RESTfulClient.java > Run As > Run on Server
* As per above program it will give you the JSON output  
  
* If you want to see the output in XML format. Open RESTfulClient.java > in line number 14 replace that line with

ClientResponse response = resource.accept(“application/xml“).get(ClientResponse.class);

and the XML output will be  
[](http://www.java4s.com/wp-content/uploads/2014/08/jax-rs-multiple-output-formats-xml-response.png)