## JAX-RS Tutorials

* [What is Web Services, Web Services Introduction](https://www.java4s.com/web-services/what-is-web-services-web-services-introduction/)
* [RESTful Web Services (JAX-RS) Annotations](https://www.java4s.com/web-services/restful-web-services-jax-rs-annotations/)
* [Jersey Hello World Example Using JAX-RS Specification](https://www.java4s.com/web-services/jersey-hello-world-example-using-jax-rs-specification/)
* [RESTful Web Services (JAX-RS) @PathParam Example](https://www.java4s.com/web-services/restful-web-services-jax-rs-pathparam-example/)
* [RESTful Web Services (JAX-RS) @QueryParam Example](https://www.java4s.com/web-services/restful-web-services-jax-rs-queryparam-example/)
* [RESTful Web Services (JAX-RS) @MatrixParam Example](https://www.java4s.com/web-services/restful-web-services-jax-rs-matrixparam-example/)
* [RESTful Web Services (JAX-RS) @FormParam Exampale](https://www.java4s.com/web-services/restful-web-services-jax-rs-formparam-example/)
* [Download Files from (JAX-RS) RESTful Web Service](https://www.java4s.com/web-services/download-files-from-jax-rs-restful-web-service/)
* [RESTful Web Service (JAX-RS) JSON Example Using Jersey](https://www.java4s.com/web-services/restful-web-service-jax-rs-json-example-using-jersey/)
* [How to Test (JAX-RS) RESTful Web Services](https://www.java4s.com/web-services/how-to-test-jax-rs-restful-web-services/)
* [JAX-RS XML Example With JAXB Using Jersey](https://www.java4s.com/web-services/jax-rs-xml-example-with-jaxb-using-jersey/)
* [RESTful Java Client Example Using Jersey Client](https://www.java4s.com/web-services/restful-java-client-example-using-jersey-client/)
* [JAX-RS Example of Multiple Resource Formats](https://www.java4s.com/web-services/jax-rs-example-of-multiple-resource-formats/)

RESTful Web Services (JAX-RS) Tutorials

[Web Services](https://www.java4s.com/web-services/) » on Jul 5, 2014 [**{ 13 Comments }**](https://www.java4s.com/web-services/restful-web-services-jax-rs-tutorials/#comments) By Sivateja

JAX-RS ( Java API for RESTful Web Services ) is a specification for RESTful Web Services with Java and it is provided by Sun.  Since it is a specification, other frameworks can be written to implement these specifications. In our JAX-RS tutorials we have used Jersey.  Just comment on these articles, In case if you have any question or issues while going through them. Enjoy learning RESTful web services in Java4s 😉

# What is Web Services, Web Services Introduction

[Web Services](https://www.java4s.com/web-services/) » on Jul 6, 2014 [**{ 23 Comments }**](https://www.java4s.com/web-services/what-is-web-services-web-services-introduction/#comments) By Sivateja

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.

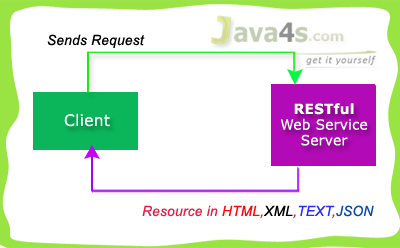
## Understanding SOAP and REST

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 timewe 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

[Web Services](https://www.java4s.com/web-services/) » on Jul 6, 2014 [**{ 8 Comments }**](https://www.java4s.com/web-services/restful-web-services-jax-rs-annotations/#comments) By Sivateja

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

[Web Services](https://www.java4s.com/web-services/) » on Jul 6, 2014 [**{ 8 Comments }**](https://www.java4s.com/web-services/how-restful-web-services-extract-input-parameters/#comments) By Sivateja

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

[Web Services](https://www.java4s.com/web-services/) » on Jul 6, 2014 [**{ 37 Comments }**](https://www.java4s.com/web-services/jersey-hello-world-example-using-jax-rs-specification/#comments) By Sivateja

In this tutorial, I will show you how to develop a RESTful hello world web application with Jersey & Maven in Eclipse.  I have used Eclipse Juno to develop all web services. Make sure you have installed Maven plugin in eclipse before you start, you can check this article for help [ [How to Install m2eclipse (Maven) Plugin in Eclipse](https://www.java4s.com/core-java/how-to-install-m2eclipse-maven-plugin-in-eclipse/) ]

[](https://www.java4s.com/wp-content/uploads/2014/07/RESTful-Hello-world-with-jersey-maven-eclipse-juno.png)

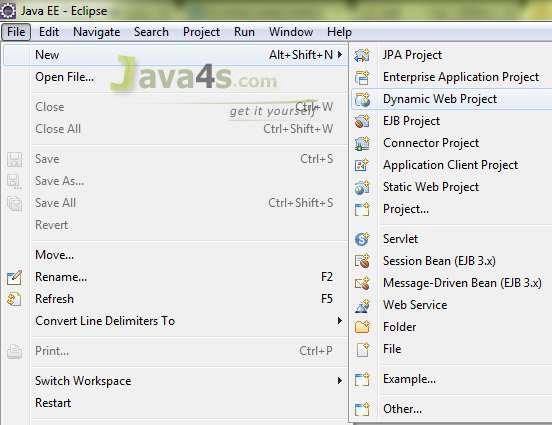
## Required

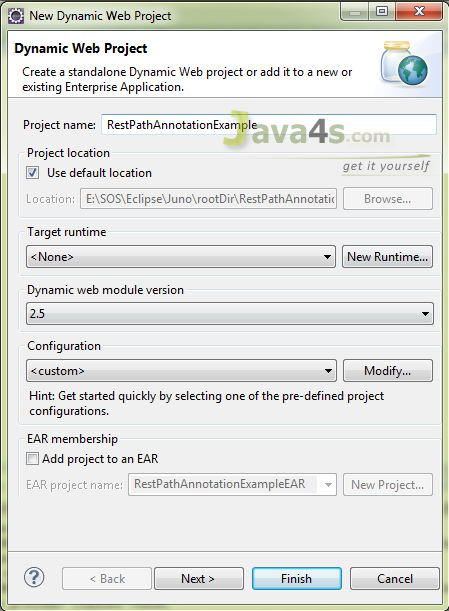
* Eclipse Juno
* JDK 1.6
* Jersey 1.8
* Maven Plugin
* Tomcat 6.0 Server

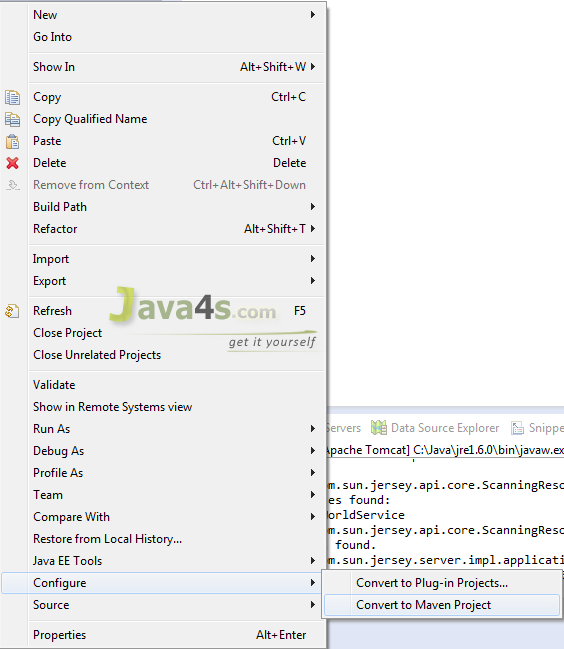
## Steps

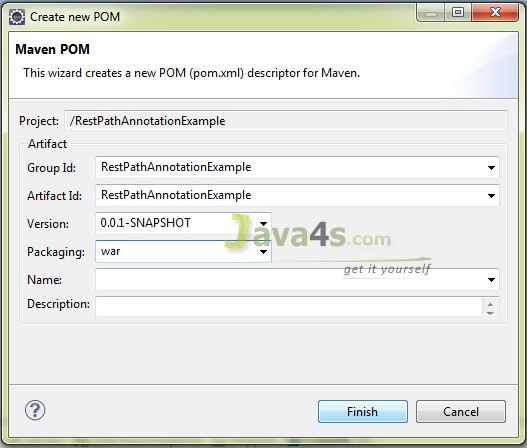
* Create a ‘Dynamic Web Project’
* Convert the project into ‘Maven Project’ [ Of course you can also create Maven project directly ]
* Add required dependencies in ‘pom.xml‘
* Change web.xml [ register com.sun.jersey.spi.container.servlet.ServletContainer and add related init-param]
* Run the application

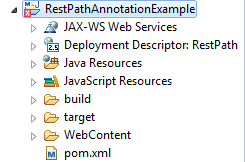
## Steps to Create Restful Web Services in Eclipse

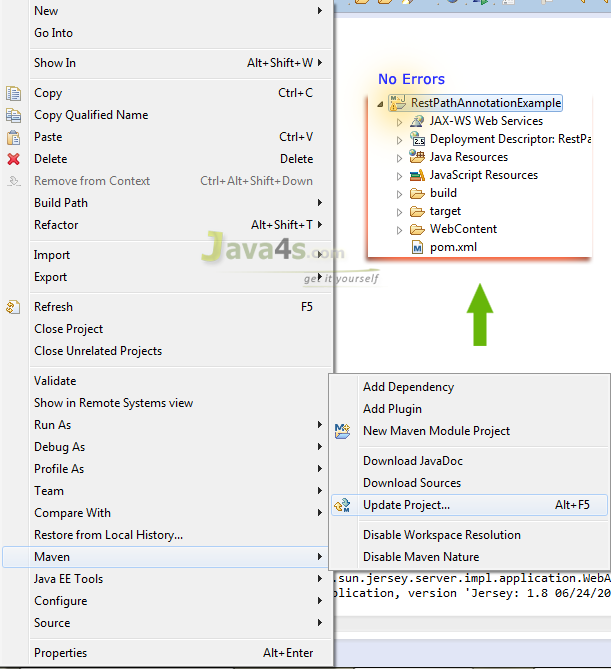
Open Eclipse > File > New > Dynamic Web Project  


Give the project name and choose ‘Dynamic web module version’ as 2.5 > Finish  


Now the project will be created in the work space, right click on the project folder > Configure > Convert to Maven Project  


Now it will open Maven POM window, there keep everything as it is, but choose packaging to .***war***and click Finish  


You have created a Maven project, finally your project looks like..  


Most probably it will not show any errors, but here Its showing errors. In order to fix this right click on the project > Maven > Update Project > Now you can see the errors gone 🙂  


Open pom.xml and add the Maven *dependencies*just like bellow

## Pom.xml

**What is pom.xml** :  
It is an XML file that contains information about the project and configuration details used by Maven to build the project.  Generally at the time of developing any J2EE applications we will search and download the related jar files over the internet and we need to add them in the class path and even in the lib folder as well. But if you can install Maven plugin in your Eclipse, pom.xml will take care of adding these dependencies (‘\*.jars) to the project. Your work is to install Maven and  update pom.xml with the required dependencies (jars). However i have already explained, how to install Maven plugin in the Eclipse.

12345678910111213141516171819202122232425262728293031323334353637383940414243444546<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>RestPathAnnotationExample</groupId>

<artifactId>RestPathAnnotationExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<repositories>

<repository>

<id>maven2-repository.java.net</id>

<name>Java.net Repository for Maven</name>

<url>http://download.java.net/maven/2/</url>

<layout>default</layout>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.8.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-server</artifactId>

<version>1.8</version>

</dependency>

</dependencies>

<build>

<finalName>RestPathAnnotationExample</finalName>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<compilerVersion>1.5</compilerVersion>

<source>1.5</source>

<target>1.5</target>

</configuration>

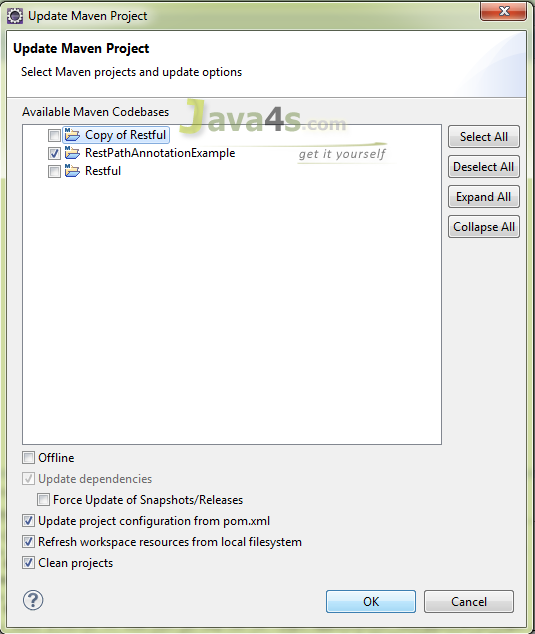
</plugin>

</plugins>

</build>

</project>

Once you add dependencies [ required libraries ] in pom.xml  > right click on the project > Maven > Update Project > Choose the current project > Ok



## HelloWorldService.java

12345678910111213141516171819202122232425262728package 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

123456789101112131415161718192021222324<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>

<!-- www.Java4s.com -->

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…

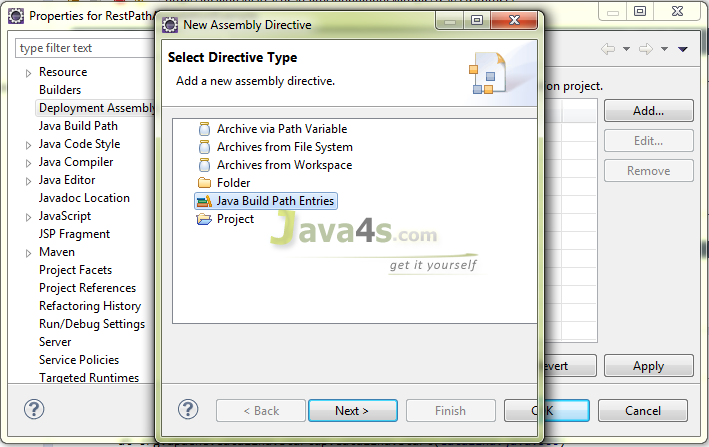
Main application URL:  
http://localhost:2013/RestPathAnnotationExample/

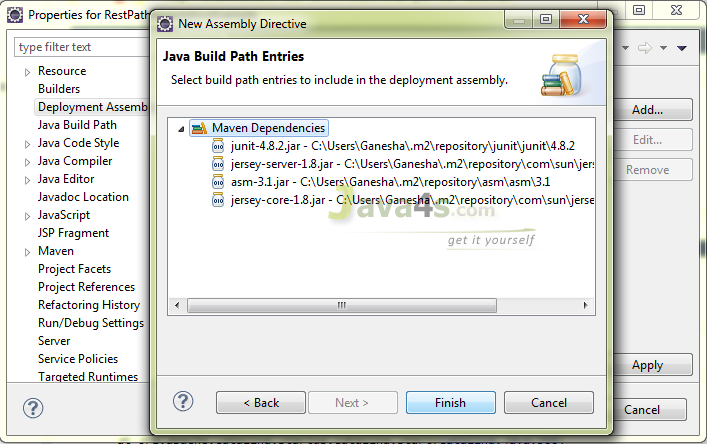
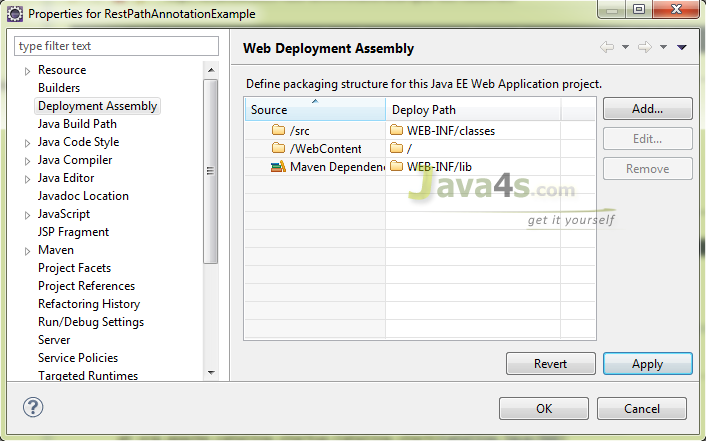
Web.xml URL pattern:  
/rest

@Path in HelloWorldService.java:  
/customers

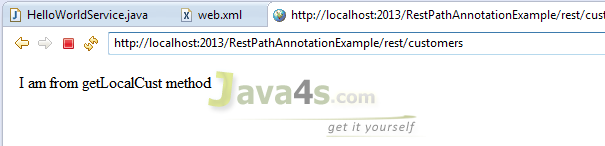
Final URL should be  
**http://localhost:2013/RestPathAnnotationExample/rest/customers**

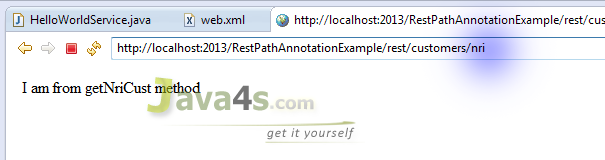
Hit with this final URL in your web browser or eclipse browser  


It will throw 404 error, because we forgot to add the Maven dependencies in the Deployment Assembly. So how to fix this issue ? just right click on the project > Properties > Deployment Assembly > Add > It will open other window, in that choose ‘Java Build Path Entries‘ click Next..  


Choose the Maven Dependencies root and Finish  
  
Now it looks like..  


Click Ok > and test the application with URLs  
<http://localhost:2013/RestPathAnnotationExample/rest/customers>

**

http://localhost:2013/RestPathAnnotationExample/rest/customers/nri  [ Here /nri  is the path i have mentioned in HelloWorldService.java  for getNriCust() method] **

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

[Web Services](https://www.java4s.com/web-services/) » on Jul 8, 2014 [**{ 9 Comments }**](https://www.java4s.com/web-services/restful-web-services-jax-rs-pathparam-example/#comments) By Sivateja

In RESTful (JAX-RS) web services @PathParam annotation will be used to bind RESTful URL parameter values to the method arguments. Lets discuss with a simple example.

**Note**:  
If you are new to RESTful web services or if you would like to know complete step by step flow of JAX-RS, Go through this article ‘[*Jersey Hello World example With Maven in Eclipse Juno*](https://www.java4s.com/web-services/jersey-hello-world-example-using-jax-rs-specification/)‘, then only you will be able to understand this tutorial 🙂 and even further web services tutorials.

## Required Files

* pom.xml
* web.xml
* RestServicePathParamJava4s.java

## pom.xml

1234567891011121314151617181920212223242526272829303132333435363738394041424344454647<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>RestPathParamAnnotationExample</groupId>

<artifactId>RestPathParamAnnotationExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<repositories>

<repository>

<id>maven2-repository.java.net</id>

<name>Java.net Repository for Maven</name>

<url>http://download.java.net/maven/2/</url>

<layout>default</layout>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.8.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-server</artifactId>

<version>1.8</version>

</dependency>

</dependencies>

<build>

<finalName>RestPathParamAnnotationExample</finalName>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<compilerVersion>1.5</compilerVersion>

<source>1.5</source>

<target>1.5</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

## web.xml

1234567891011121314151617181920212223<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>RestPathParamAnnotationExample</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>

<!-- www.Java4s.com -->

## RestServicePathParamJava4s.java

1234567891011121314151617181920212223package 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();

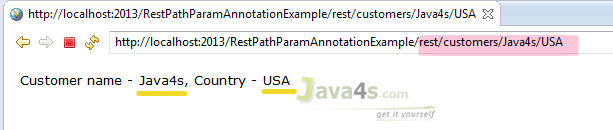
}

}

## 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, Stringcountry respectively.  
  
* Check the output

## Output



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

[Web Services](https://www.java4s.com/web-services/) » on Jul 8, 2014 [**{ 6 Comments }**](https://www.java4s.com/web-services/restful-web-services-jax-rs-queryparam-example/#comments) By Sivateja

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](https://www.java4s.com/hibernate/restful-web-services-jax-rs-pathparam-example/) no changes 🙂
* RestServiceQueryParamJava4s.java

## RestServiceQueryParamJava4s.java

12345678910111213141516171819202122package com.java4s;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.QueryParam;

import javax.ws.rs.core.Response;

*@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();

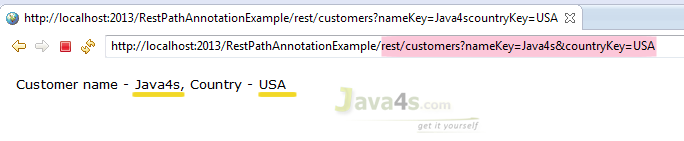
}

}

## 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

[Web Services](https://www.java4s.com/web-services/) » on Jul 8, 2014 [**{ 4 Comments }**](https://www.java4s.com/web-services/restful-web-services-jax-rs-matrixparam-example/#comments) By Sivateja

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](https://www.java4s.com/web-services/restful-web-services-jax-rs-pathparam-example/)]
* RestServiceMatrixParamJava4s.java

## RestServiceMatrixParamJava4s.java

12345678910111213141516171819202122package com.java4s;

import javax.ws.rs.GET;

import javax.ws.rs.MatrixParam;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.Response;

*@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();

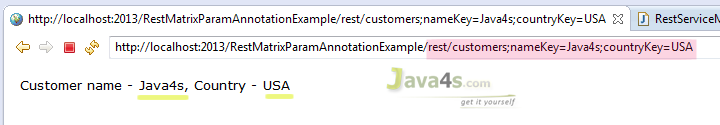
}

}

## 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

[Web Services](https://www.java4s.com/web-services/) » on Jul 9, 2014 [**{ 3 Comments }**](https://www.java4s.com/web-services/restful-web-services-jax-rs-formparam-example/#comments) By Sivateja

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.

Let me give you an example on usage of @FormParam in the JAX-RS.

**Note**:  
If you are new to RESTful web services, first go through ‘[*Jersey Hello World Example Using JAX-RS Specification*](https://www.java4s.com/web-services/jersey-hello-world-example-using-jax-rs-specification/)‘ there you can learn each and every step to create a RESTful web service in eclipse, how to install maven and configuration settings related to JAX-RS.

Required Files

* pom.xml and web.xml are similar to the [previous article](https://www.java4s.com/web-services/jersey-hello-world-example-using-jax-rs-specification/)
* RestServiceFormParamJava4s.java
* Client.html

## RestServiceFormParamJava4s.java

123456789101112131415161718192021222324package com.java4s;

import javax.ws.rs.FormParam;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.Response;

*@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

1234567891011121314151617181920212223242526272829<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>

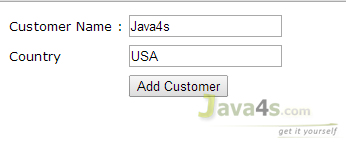
## Explanation

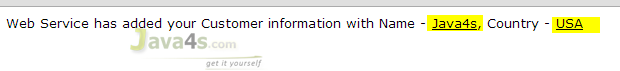
* Right click on your project root folder > Run As > Run on Server
* Eclipse will open http://localhost:2013/<projectRootFolder> with 404 Error by default, forget about that
* Now open Client.html in your web browser, enter the details and click submit [ I have created this .html file to send input form 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:  


# Download Files from (JAX-RS) RESTful Web Service

[Web Services](https://www.java4s.com/web-services/) » on Jul 10, 2014 [**{ 6 Comments }**](https://www.java4s.com/web-services/download-files-from-jax-rs-restful-web-service/#comments) By Sivateja

In this article i will show you how to download files from your JAX-RS  web service.  Downloading files from restful is easier compared to upload :-), however i will give you both examples.  We can download any type of files from the RESTful web services, its just a matter of changing @produces annotation. For example..

We should annotate our method with

@Produces(“text/plain“) If you are expecting Text file as response  
@Produces(“image/your image type[.jpg/.png/.gif]”) for downloading any Image files  
@Produces(“application/pdf“) for downloading PDF files

Lets discuss these three scenarios with an example.

Required Files

* pom.xml & web.xml [ Refer this [Restful Hello world example](https://www.java4s.com/web-services/jersey-hello-world-example-using-jax-rs-specification/), i am using the same xml’s ]
* RestServiceFileDownloadJava4s.java

## RestServiceFileDownloadJava4s.java

123456789101112131415161718192021222324252627282930313233package com.java4s;

import java.io.File;import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.Response;

import javax.ws.rs.core.Response.ResponseBuilder;

*@Path("/downnload")*

public class RestServiceFileDownloadJava4s {

String path = "c:\\tuts\\java4s.txt";

/\* public File getCustomerDataFile() {

File file = new File(path);

return file;

}\*/

*@GET*

*@Path("/data")*

*@Produces("text/plain")*

//*@Produces("image/png")*

//*@Produces("application/pdf")*

public Response getCustomerDataFile() {

File file = new File(path);

ResponseBuilder rb = Response.ok((Object) file);

rb.header("Content-Disposition","attachment; filename=java4sFileFromServer.txt");

return rb.build();

}

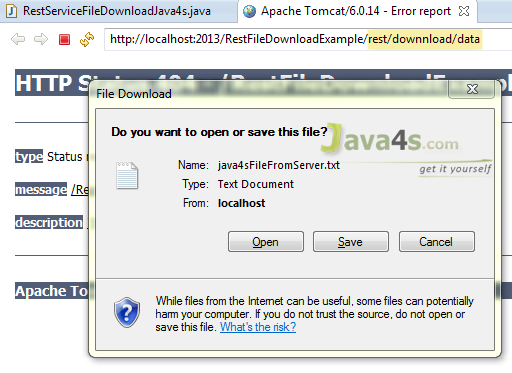
}

## Explanation

* Our intention is to download the TEXT file from JAX-RS, for that we need to annotate our method with @Produces(“text/plain”) [which i did in line number 24]
* Once we call the RESTful service, i want to display a pop-up download box for the users to ‘download‘ that file, in order to do that we need to add ‘Content-Disposition‘ header to the response
* But in the Response class we don’t have any option to add the headers, so firstly i have created ‘ResponseBuilder‘ object [ line number 31, because in ResponseBuilder class we have direct method to add the headers], and added ‘**Content-Disposition**‘ to the header.
* Finally called rb.build() [at line number 33], this will create a Response instance from the current ResponseBuilder object (rb) and returns
* We can also get the output by simply writing the lines 16-19 but it wont shows download pop-up box 🙂
* You can enable, line numbers 25,26 if your file is Image & PDF respectively

Same thing will happen in case of **Images/PDF** or other file formats.

## Output



# RESTful Web Service (JAX-RS) JSON Example Using Jersey

[Web Services](https://www.java4s.com/web-services/) » on Jul 19, 2014 [**{ 17 Comments }**](https://www.java4s.com/web-services/restful-web-service-jax-rs-json-example-using-jersey/#comments) By Sivateja

This article describes how to get a JSON response from the RESTful web services using jerseyimplementation.  Jersey will use Jackson to convert Java objects to/form JSON, but just don’t ask me what is Jackson 🙂 ,as of now just remember its a high performance JSON processor, Jersey will use this API to the marshaling [converting the objects] process. Check this link if you would like to know more about Jackson.

These steps are mandatory in order to make Jersey to support with JSON mappings.

* Apart from existing dependencies, add ‘jersey-json.jar‘ to your Maven pom.xml which includes all Jackson and other JSON supporting libraries
* 12345 <dependency>
* <groupId>com.sun.jersey</groupId>
* <artifactId>jersey-json</artifactId>
* <version>1.8</version>

</dependency>

* In web.xml add “com.sun.jersey.api.json.POJOMappingFeature” as “init-param”
* In the web service class, we need to annotate the method with @Produces(MediaType.APPLICATION\_JSON). By doing so we are instructiong the service method that we are expecting the JSON output, thats it jersey will take care rest of the things

**Note**: In previous examples i used Tomcat 6 and JDK 1.6 but for this [JAX-RS JSON Example] example i have used JDK 1.7.

## Required Files

* pom.xml
* web.xml
* Customer.java
* JsonFromRestful.java

## pom.xml

12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>JsonFromRestfulWebServices</groupId>

<artifactId>JsonFromRestfulWebServices</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<repositories>

<repository>

<id>maven2-repository.java.net</id>

<name>Java.net Repository for Maven</name>

<url>http://download.java.net/maven/2/</url>

<layout>default</layout>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.8.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-server</artifactId>

<version>1.8</version>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-json</artifactId>

<version>1.8</version>

</dependency>

</dependencies>

<build>

<finalName>JsonFromRestfulWebServices</finalName>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<compilerVersion>1.5</compilerVersion>

<source>1.5</source>

<target>1.5</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

## web.xml

1234567891011121314151617181920212223<?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>JsonFromRestfulWebServices</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

123456789101112131415161718192021222324252627package java4s;

public class Customer {

private int custNo;

private String custName;

private String custCountry;

public int getCustNo() {

return custNo;

}

public void setCustNo(int custNo) {

this.custNo = custNo;

}

public String getCustName() {

return custName;

}

public void setCustName(String custName) {

this.custName = custName;

}

public String getCustCountry() {

return custCountry;

}

public void setCustCountry(String custCountry) {

this.custCountry = custCountry;

}

}

## JsonFromRestful.java

1234567891011121314151617181920212223242526272829package 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.MediaType;

*@Path("/customers")*

public class JsonFromRestful {

*@GET*

*@Path("/{cusNo}")*

*@Produces(MediaType.APPLICATION\_JSON)*

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

/\*

\* I AM PASSING CUST.NO AS AN INPUT, SO WRITE SOME BACKEND RELATED STUFF AND

\* FIND THE CUSTOMER DETAILS WITH THAT ID. AND FINALLY SET THOSE RETRIEVED VALUES TO

\* THE CUSTOMER OBJECT AND RETURN IT, HOWEVER IT WILL RETURN IN JSON FORMAT 🙂

\* \*/

Customer cust = new Customer();

cust .setCustNo(no);

cust .setCustName("Java4s");

cust .setCustCountry("India");

return cust;

}

}

## Output



# How to Test (JAX-RS) RESTful Web Services

[Web Services](https://www.java4s.com/web-services/) » on Jul 20, 2014 [{ 1 Comment }](https://www.java4s.com/web-services/how-to-test-jax-rs-restful-web-services/#comments) By Sivateja

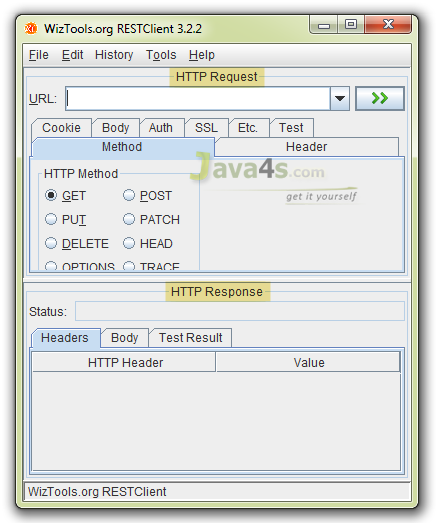
In this article i will show you how to test RESTful web service (JAX-RS), so far we have learnedhow to create a RESTful service and testing GET and POST requests through some web browser.  But in real time projects we will use different tools to test RESTful web services.  If you would like to test JAX-RS with web browser you can use the following tools…

* Postman [ Chrome Extension ]
* REST Client [ Chrome Extension ]
* Advanced REST Client [ Chrome Extension ]
* Rest Client [ Firefox Add-On ]

If you would like to test JAX-RS in your local

* RESTClient UI
* SoupUi

In this tutorial i will show you how to test jax-rs with RESTClient UI

* Click here to Download [RESTClient UI](https://code.google.com/p/rest-client/downloads/list" \t "_blank)
* Open the above link and download restclient-ui-3.2.2-jar-with-dependencies.jar
* We are done, you no need to do any configurations kind of things, Just double click on the downloaded jar file to run the application, it looks like…  
  

Lets take an example with GET, POST, PUT, DELETE for testing the web service.

## Required Files

* pom.xml
* web.xml
* TestingRestfulWebService.java

## pom.xml

1234567891011121314151617181920212223242526272829303132333435363738394041424344454647<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>TestRestfulWebServiceExample</groupId>

<artifactId>TestRestfulWebServiceExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<repositories>

<repository>

<id>maven2-repository.java.net</id>

<name>Java.net Repository for Maven</name>

<url>http://download.java.net/maven/2/</url>

<layout>default</layout>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.8.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-server</artifactId>

<version>1.8</version>

</dependency>

</dependencies>

<build>

<finalName>TestRestfulWebServiceExample</finalName>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<compilerVersion>1.5</compilerVersion>

<source>1.5</source>

<target>1.5</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

## web.xml

12345678910111213141516171819202122<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>TestRestfulWebServiceExample</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>

## TestingRestfulWebService.java

12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455package com.java4s;

import javax.ws.rs.DELETE;

import javax.ws.rs.GET;

import javax.ws.rs.POST;

import javax.ws.rs.PUT;

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 TestingRestfulWebService {

*@GET*

*@Produces("text/plain")*

*@Path("{id}")*

public Response getCustomerDetails(*@PathParam("id") String custId) {*

//CODE TO FETCH CUSTOMER DETAILS FROM THE DATABASE USING CUSTOMER ID

String output = "Customer Details With ID "+custId+" Has Been fetched Successfully";

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

}

*@POST*

*@Produces("text/plain")*

*@Path("{id}")*

public Response insertCustomer(*@PathParam("id") String custId) {*

//CODE TO INSERT CUSTOMER DETAILS USING CUSTOMER ID

String output = "Customer Data With ID "+custId+" Has Been Saved Successfully";

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

}

*@PUT*

*@Produces("text/plain")*

*@Path("{id}")*

public Response updateCustomerDetails(*@PathParam("id") String custId) {*

//CODE TO UPDATE CUSTOMER DETAILS USING CUSTOMER ID

String output = "Customer Data With ID "+custId+" Has Been Updated Successfully";

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

}

*@DELETE*

*@Produces("text/plain")*

*@Path("{id}")*

public Response deleteCustomer(*@PathParam("id") String custId) {*

//CODE TO DELETE CUSTOMER DETAILS USING CUSTOMER ID

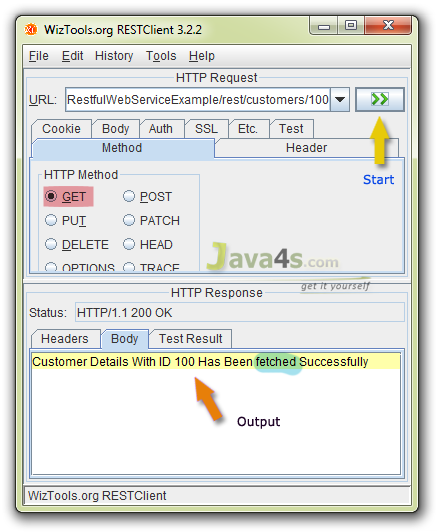
String output = "Customer With ID "+custId+" Has Been Deleted From the Database.";

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

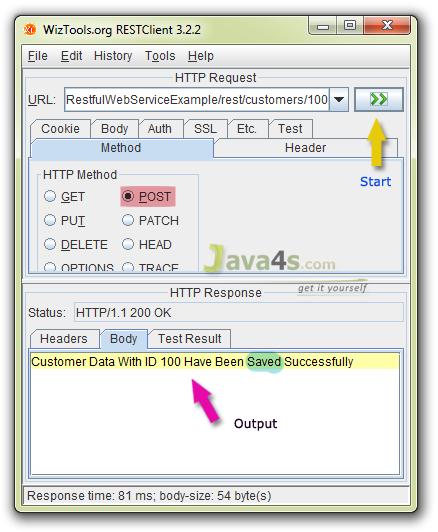
}

}

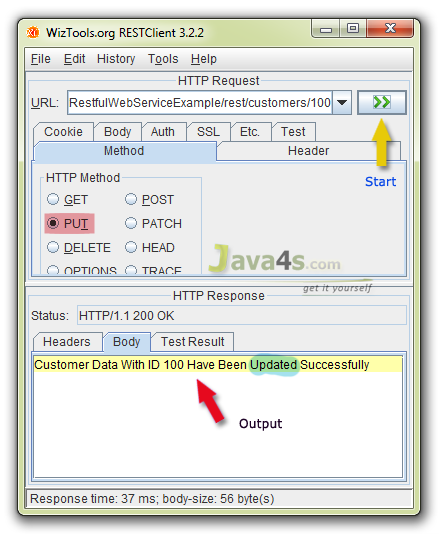
## Testing JAX-RS GET Request

* Eclipse > Run the application > Now open Restclient UI
* In the URL field enter  http://localhost:2013/TestRestfulWebServiceExample/rest/customers/100
* Choose GET method in the ‘HTTP Method’ options > now hit the start button and check the output
* **Output**  
  

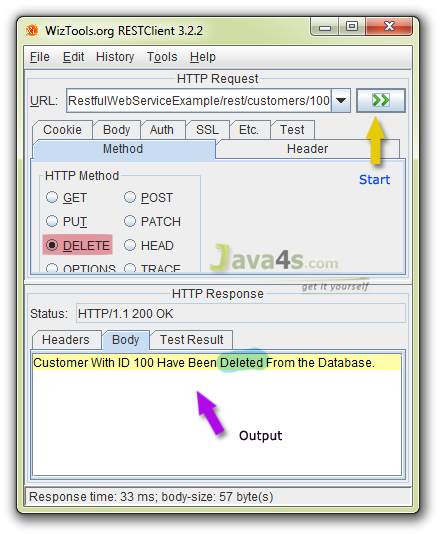
## Testing JAX-RS POST Request

* Choose POST method in the ‘HTTP Method’ options > Hit the start button and check the output
* Output  
  

## Testing JAX-RS PUT Request

* Choose PUT method in the ‘HTTP Method’ options > Hit the start button and check the output
* Output  
  

## Testing JAX-RS DELETE Request

* Choose DELETE method in the ‘HTTP Method’ options > Hit the start button and check the output
* Output  
  

# JAX-RS XML Example With JAXB Using Jersey

[Web Services](https://www.java4s.com/web-services/) » on Jul 25, 2014 [**{ 7 Comments }**](https://www.java4s.com/web-services/jax-rs-xml-example-with-jaxb-using-jersey/#comments) By Sivateja

In this article i will give you an example on how a RESTful web service produces XML response using Jersey. Basically JAX-RS supports conversion of java objects into XML with the help of JAXB. As Jersey it self contains JAXB libraries we no need to worry about JAXB-Jersey integrationstuff.

## Steps Need to be Followed

* Add ‘jersey-server.jar‘ to your Maven pom.xml which includes all JAXB supporting libraries into your class path
* Annotate your service method with @Produces(MediaType.APPLICATION\_XML)

## Required Files

* pom.xml
* web.xml
* Customer.java
* RestfulXMLExample.java

## pom.xml

1234567891011121314151617181920212223242526272829303132333435363738394041424344454647<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>RestfulXMLExample</groupId>

<artifactId>RestfulXMLExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<repositories>

<repository>

<id>maven2-repository.java.net</id>

<name>Java.net Repository for Maven</name>

<url>http://download.java.net/maven/2/</url>

<layout>default</layout>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.8.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-server</artifactId>

<version>1.8</version>

</dependency>

</dependencies>

<build>

<finalName>RestfulXMLExample</finalName>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<compilerVersion>1.5</compilerVersion>

<source>1.5</source>

<target>1.5</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

## web.xml

12345678910111213141516171819202122<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>

## Customer.java

12345678910111213141516171819202122232425262728293031323334353637package com.java4s;

import javax.xml.bind.annotation.XmlAttribute;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

*@XmlRootElement(name = "customer")*

public class Customer {

String custName;

String custCountry;

int custId;

*@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;

}

*@XmlAttribute*

public int getCustId() {

return custId;

}

public void setCustId(int custId) {

this.custId = custId;

}

}

## RestfulXMLExample.java

123456789101112131415161718192021222324252627package 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.MediaType;

*@Path("/customers")*

public class RestfulXMLExample {

*@GET*

*@Path("/{id}")*

*@Produces(MediaType.APPLICATION\_XML)*

public Customer getCustomerDetails(*@PathParam("id") int custId) {*

// WRITE DATABASE LOGIC TO RETRIEVE THE CUSTOMER RECORD WITH 'custID'

Customer cust = new Customer();

cust.setCustName("Java4s");

cust.setCustCountry("USA");

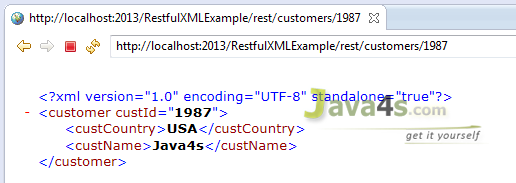
cust.setCustId(custId);

return cust;

}

}

## Output

[](https://www.java4s.com/wp-content/uploads/2014/07/jax-rs-xml-example-jersey.png)

# RESTful Java Client Example Using Jersey Client

[Web Services](https://www.java4s.com/web-services/) » on Aug 5, 2014 [**{ 11 Comments }**](https://www.java4s.com/web-services/restful-java-client-example-using-jersey-client/#comments) By Sivateja

In this article i will describe how to write a JAX-RS client application using jersey client API, so far we used to call & test/read our RESTful service by its URL directly hitting in the browser [ check the previous examples ], but in the real time we will call the services by writing some client application logic.  We have different ways to write a RESTful client.  In this article i will use Jersey client [ of course we are using Jersey for writing RESTful service too 🙂 ]

## Steps need to be followed

* Need to add ‘jersey-client‘ dependency in pom.xml
* As we are creating the Client application, we need to write a RESTful service to test that client, so i will take previous [JSON example](https://www.java4s.com/web-services/restful-web-service-jax-rs-json-example-using-jersey/) in order to do that
* Write a client application and run it 🙂

## Files Required

* pom.xml
* web.xml
* JsonFromRestful.java
* Customer.java
* RESTfulClient.java

## pom.xml

In the pom.xml we must include ‘jersey-client’ dependency [ Check line numbers from 4-10 ]

1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>JAX-RS-Client-Example</groupId>

<artifactId>JAX-RS-Client-Example</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<repositories>

<repository>

<id>maven2-repository.java.net</id>

<name>Java.net Repository for Maven</name>

<url>http://download.java.net/maven/2/</url>

<layout>default</layout>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.8.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-server</artifactId>

<version>1.8</version>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-json</artifactId>

<version>1.8</version>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-client</artifactId>

<version>1.8</version>

</dependency>

</dependencies>

<build>

<finalName>JAX-RS-Client-Example</finalName>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<compilerVersion>1.5</compilerVersion>

<source>1.5</source>

<target>1.5</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

## web.xml

12345678910111213141516171819202122<?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-Client-Example</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>

## JsonFromRestful.java

12345678910111213141516171819202122232425262728package 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/json")*

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

/\*

\* I AM PASSING CUST.NO AS AN INPUT, SO WRITE SOME BACKEND RELATED STUFF AND

\* FIND THE CUSTOMER DETAILS WITH THAT ID. AND FINALLY SET THOSE RETRIEVED VALUES TO

\* THE CUSTOMER OBJECT AND RETURN IT, HOWEVER IT WILL RETURN IN JSON FORMAT 🙂

\* \*/

Customer cust = new Customer();

cust .setCustNo(no);

cust .setCustName("Java4s");

cust .setCustCountry("India");

return cust;

}

}

## Customer.java

12345678910111213141516171819202122232425262728package java4s;

public class Customer {

private int custNo;

private String custName;

private String custCountry;

public int getCustNo() {

return custNo;

}

public void setCustNo(int custNo) {

this.custNo = custNo;

}

public String getCustName() {

return custName;

}

public void setCustName(String custName) {

this.custName = custName;

}

public String getCustCountry() {

return custCountry;

}

public void setCustCountry(String custCountry) {

this.custCountry = custCountry;

}

}

## RESTfulClient.java

12345678910111213141516171819202122232425262728package 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-Client-Example/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();

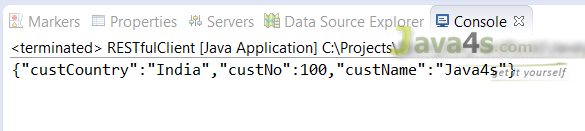
}

}

}

## Explanation:

* Line number 13, we are calling our rest service
* Line number 14, we are intimating the REST Service that our client will accept JSON response
* Right click on RESTfulClient.java in eclipse > Run As > Run on Server



# JAX-RS Example of Multiple Resource Formats

[Web Services](https://www.java4s.com/web-services/) » on Aug 6, 2014 [**{ 24 Comments }**](https://www.java4s.com/web-services/jax-rs-example-of-multiple-resource-formats/#comments) By Sivateja

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.

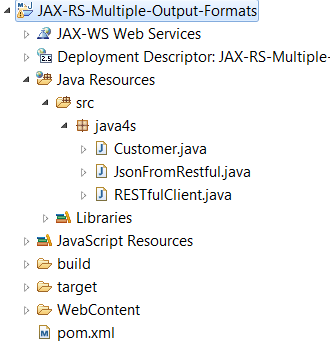
* [RESTful Web Service (JAX-RS) JSON Example Using Jersey](https://www.java4s.com/web-services/restful-web-service-jax-rs-json-example-using-jersey/)
* [JAX-RS XML Example With JAXB Using Jersey](https://www.java4s.com/web-services/jax-rs-xml-example-with-jaxb-using-jersey/)

Exactly, exactly this current example is the combination of above two articles 🙂 I am just copy & pasting those articles and doing some changes here that’s it.

## Required Files

* pom.xml
* web.xml
* Customer.java
* JsonFromRestful.java
* RESTfulClient.java

## Directory Structure



## pom.xml

123456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354555657<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>JAX-RS-Multiple-Output-Formats</groupId>

<artifactId>JAX-RS-Multiple-Output-Formats</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<repositories>

<repository>

<id>maven2-repository.java.net</id>

<name>Java.net Repository for Maven</name>

<url>http://download.java.net/maven/2/</url>

<layout>default</layout>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.8.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-server</artifactId>

<version>1.8</version>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-json</artifactId>

<version>1.8</version>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-client</artifactId>

<version>1.8</version>

</dependency>

</dependencies>

<build>

<finalName>JAX-RS-Multiple-Output-Formats</finalName>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<compilerVersion>1.5</compilerVersion>

<source>1.5</source>

<target>1.5</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

## web.xml

1234567891011121314151617181920212223<?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

1234567891011121314151617181920212223242526272829303132333435363738package 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

1234567891011121314151617181920212223package 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

12345678910111213141516171819202122232425262728package 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) {

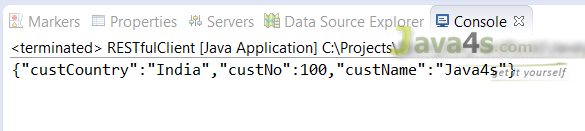
e.printStackTrace();

}

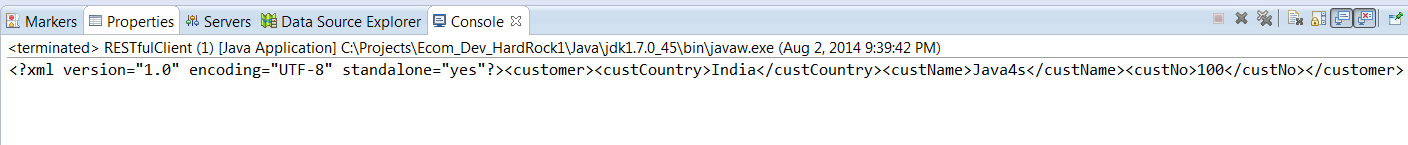
}

}

## 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

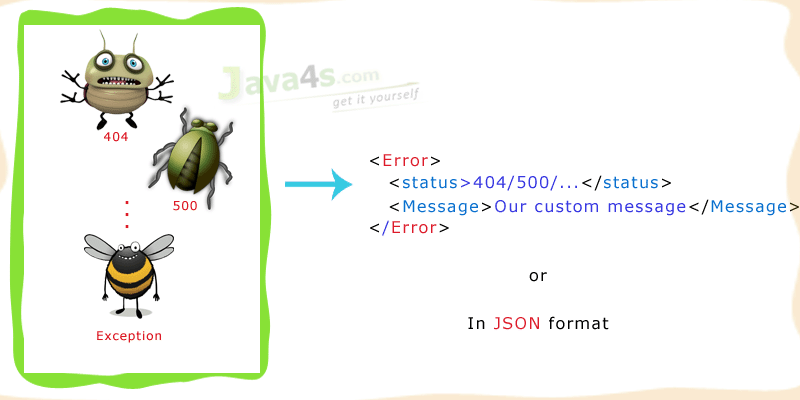
ClientResponse response = resource.accept(“application/xml“).get(ClientResponse.class);

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

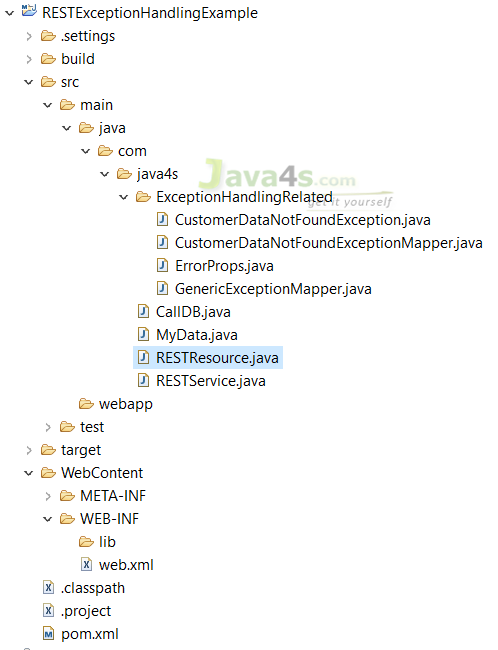
# Exception Handling in RESTful Web Services (JAX-RS) with Jersey

[Web Services](https://www.java4s.com/web-services/) » on May 21, 2017 [**{ 13 Comments }**](https://www.java4s.com/web-services/exception-handling-in-restful-web-services-jax-rs-with-jersey/#comments) By Sivateja

Exception handling in RESTful (JAX-RS) web services is a very important concept, in this article I am going to explain it step by step with an example. FYI. check this article for [Creating Simple Maven RESTful Web Service Project in Eclipse](https://www.java4s.com/web-services/jersey-hello-world-example-using-jax-rs-specification/). I am directly going to start with directory structure of the current example.



## Directory structure



As we are dealing with exception handling, I am going to create a service which will throw an exception 🙂

## pom.xml

123456789101112131415161718192021222324252627282930313233343536373839404142434445464748<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>RESTExceptionHandlingExample</groupId>

<artifactId>RESTExceptionHandlingExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<repositories>

<repository>

<id>maven2-repository.java.net</id>

<name>Java.net Repository for Maven</name>

<url>http://download.java.net/maven/2/</url>

<layout>default</layout>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.8.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-server</artifactId>

<version>1.8</version>

</dependency>

</dependencies>

<build>

<finalName>RESTExceptionHandlingExample</finalName>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<compilerVersion>1.8</compilerVersion>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

## web.xml

1234567891011121314151617181920<?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:web="http://xmlns.jcp.org/xml/ns/javaee"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd http://xmlns.jcp.org/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID" version="2.4">

<display-name>RESTExceptionHandlingExample</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>/\*</url-pattern>

</servlet-mapping>

</web-app>

## RESTResource.java

123456789101112131415161718192021package 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.MediaType;

import javax.ws.rs.core.Response;

*@Path("/customers")*

public class RESTResource {

*@GET*

*@Path("/checkProfile/{id}")*

public Response getAdminDetails(*@PathParam("id") String id) {*

String msg = RESTService.checkCustomerStatus(id);

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

}

}

## RESTService.java

12345678910111213package com.java4s;

public class RESTService {

CallDB cdb = new CallDB();

public String checkCustomerStatus(String custId){

MyData da = cdb.getStatus(custId);

return da.getStatus().trim();

}

}

## CallDB.java

1234567891011121314151617package com.java4s;

public class CallDB {

public MyData getStatus(String custId) {

// Lets say, database logic will go here and setting the output in MyData bean

MyData da = new MyData();

// da.setStatus("Valid");

return da;

}

}

## MyData.java

1234567891011121314package com.java4s;

public class MyData {

String status;

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

}

## Explanation

So we are good to run the application now,  if you observe I have just created 4 java classes till now..

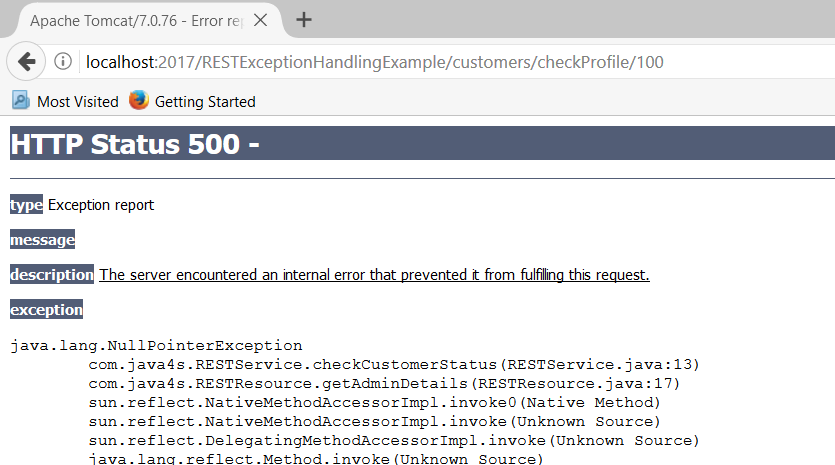
* RESTResource.java
* RESTService.java
* CallDB.java
* MyData.java

The application will start by hitting the following URL [ I am using 2017 as my server port, this might be different for your server]

http://localhost:2017/RESTExceptionHandlingExample/customers/checkProfile/100

* If you hit the above URL, the flow will come to RESTResource.java, in that at line number 17, I am calling checkCustomerStatus( – ) function of RESTService class, by passing the id
* Consider, CallDB is a class which handle all database related stuff, so in RESTService.java > line number 11, I am calling getStatus( – ) of CallDB
* In CallDB.java > consider we had a database call at line number 7,  after that creating MyData class object and setting all the data that I retrieved from the database and returning it at line number 13, if you observe line number 11, I have commented the setter method, means I am not setting status, so by default it contains NULL value ( if we call its getter method, it will return NULL)
* Now the flow will come to RESTService.java > at line number 11, I am calling getStatus(), as this gives null value **[**as we have not set anything in CallDB > line 11**]**, on that null I am calling .trim() again, which will give NullPointerException 🙂

Hmm.. so successfully we are able to create a service, which will throw a NullPointerException 🙂 If you hit the URL, you will see..

  
Its the Tomcat default error page, showing the exception as NullPointerException but I don’t think its the right way of displaying the errors to the consumers! lets try to display the error with some custom error message, in order to do that I am going to create a custom (user defined) run time exception, lets say my custom exception class name is CustomerDataNotFoundException.

## CustomerDataNotFoundException.java

123456789101112package com.java4s.ExceptionHandlingRelated;

public class CustomerDataNotFoundException extends RuntimeException{

private static final long serialVersionUID = 1L;

public CustomerDataNotFoundException(String exceptionMsg)

{

super(exceptionMsg);

}

}

Here I have just created an exception class with constructor which takes String as an argument, now let me change RESTService.java as..

## RESTService.java

123456789101112131415161718192021package com.java4s;

import com.java4s.ExceptionHandlingRelated.CustomerDataNotFoundException;

public class RESTService {

CallDB cdb = new CallDB();

public String checkCustomerStatus(String custId){

MyData da = cdb.getStatus(custId);

if(da.getStatus() == null)

{

throw new CustomerDataNotFoundException("Customer status not found with id "+custId);

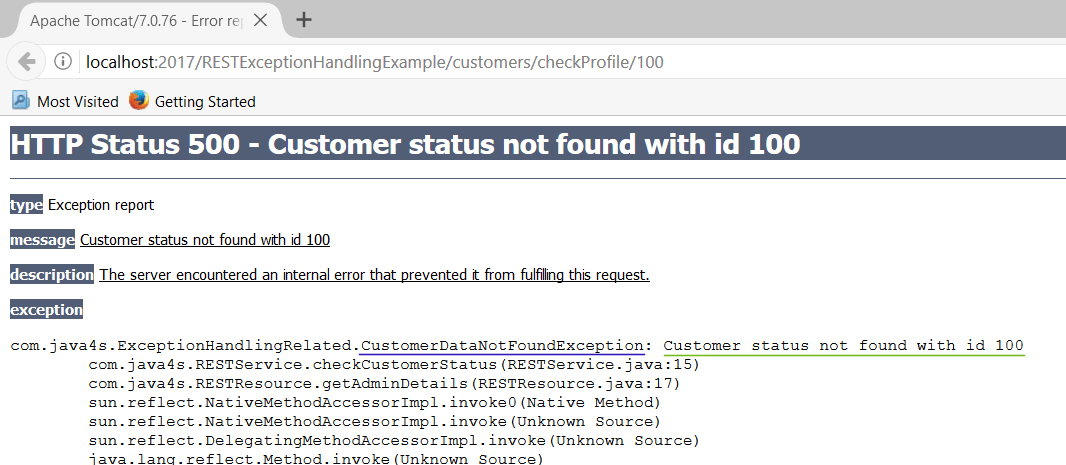
}

return da.getStatus().trim();

}

}

I am checking the status with if condition. If status is NULL, throwing out the custom exception by passing some meaningful message. Lets run the application and see..

[](https://www.java4s.com/wp-content/uploads/2017/05/jax-rs-exceptions-example.gif)  
Its throwing our exception with the message we have sent, not bad 😉 but still this is not the right way of showing the errors.

## What happens behind the scenes

Actually we are throwing CustomerDataNotFoundException if the status is NULL, if you observe, we are not handling that exception in RESTService, instead simply throwing with our message. So the exception keeps bubbling up and will come to RESTResource (here also we are not handling ) and so from there to JAX-RS and finally will reach Tomcat server container, and server will show its default error page, that’s what we are seeing in the above image.

So in order to stop exception bubbling up to the tomcat server container, we need to create an exception mapper.

## CustomerDataNotFoundExceptionMapper.java

1234567891011121314151617package com.java4s.ExceptionHandlingRelated;

import javax.ws.rs.core.Response;

import javax.ws.rs.core.Response.Status;

import javax.ws.rs.ext.ExceptionMapper;

import javax.ws.rs.ext.Provider;

*@Provider*

public class CustomerDataNotFoundExceptionMapper implements ExceptionMapper<CustomerDataNotFoundException>{

public Response toResponse(CustomerDataNotFoundException ex)

{

return Response.status(Status.NOT\_FOUND)

.entity(new ErrorProps("404", ex.getMessage()))

.build();

}

}

## ErrorProps.java

12345678910111213141516171819202122232425262728293031323334package com.java4s.ExceptionHandlingRelated;

import javax.xml.bind.annotation.XmlRootElement;

*@XmlRootElement*

public class ErrorProps {

private String status;

private String errorMessage;

public ErrorProps(){}

public ErrorProps(String statusFromOutside, String errorMessageFromOutside)

{

this.status = statusFromOutside;

this.errorMessage = errorMessageFromOutside;

}

public String getErrorMessage() {

return errorMessage;

}

public void setErrorMessage(String errorMessage) {

this.errorMessage = errorMessage;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

}

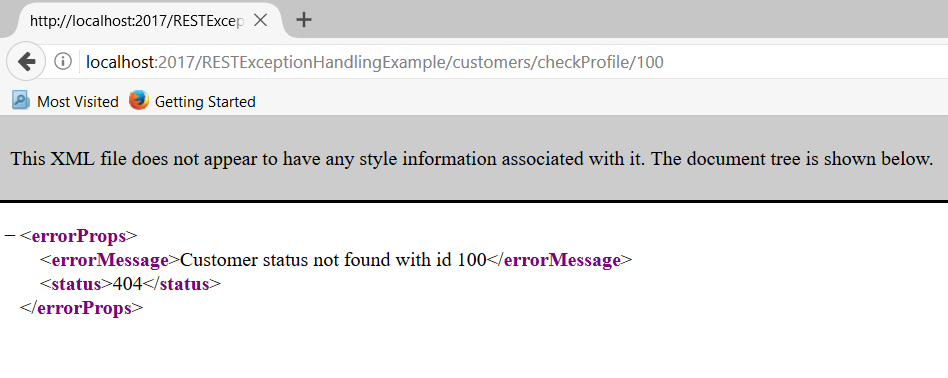
## Explanation

* Created an exception mapper CustomerDataNotFoundExceptionMapper for CustomerDataNotFoundException
* All exception mappers should implement ExceptionMapper interface of type generic, for now I am going to use this exception mapper only for our exception, so I have implemented ExceptionMapper of type CustomerDataNotFoundException [check at line number 9]
* We need to override the toResponse method of ExceptionMapper interface, which takes exception as an argument, in this case CustomerDataNotFoundException
* I want to display my exception details as an XML, so created a simple java model ErrorProps.java and annotated with @XmlRootElement
* Now come back to mapper class toResponse method, there I am returning Response object

Response.status( – ) :- setting the current status  
.entity( – ) :- passing ErrorProps class object by setting required values, here I am setting status as 404, and our custom exception message

* Finally annotate our mapper class with @Provider annotation, so that JAX-RS will register this mapper to intercept the response when particular exception was thrown

Go ahead and run the application and see…

[](https://www.java4s.com/wp-content/uploads/2017/05/jax-rs-exceptions-in-xml.PNG)

We did it 🙂

But this is only for NullPointerException, but how about the other exceptions ? for that we need to modify the mapper. Let me do it by creating new mapper class.

## GenericExceptionMapper.java

12345678910111213141516171819202122232425262728package com.java4s.ExceptionHandlingRelated;

import javax.ws.rs.core.Response;

import javax.ws.rs.core.Response.Status;

import javax.ws.rs.ext.ExceptionMapper;

import javax.ws.rs.ext.Provider;

*@Provider*

public class GenericExceptionMapper implements ExceptionMapper<Throwable>{

public Response toResponse(Throwable ex)

{

if(ex instanceof CustomerDataNotFoundException)

{

return Response.status(Status.NOT\_FOUND)

.entity(new ErrorProps("404", ex.getMessage()))

.build();

}

else

{

return Response.status(Status.INTERNAL\_SERVER\_ERROR)

.entity(new ErrorProps("Some error code, 500 or somthing", ex.getMessage()))

.build();

}

}

}

## What are the changes ?

Implement ExceptionMapper of type Throwable, instead of our own exception. If you check the above class, at line number 13, I am checking whether the Throwable is the instance of CustomerDataNotFoundException, if its true, I will build my Response accordingly, like this you can handle any number of exceptions in a single class, you can download and play with it.

[[https://www.java4s.com/wp-content/uploads/2011/05/download.png](https://www.java4s.com/wp-content/web-services/RESTExceptionHandlingExample.rar)](https://www.java4s.com/wp-content/web-services/RESTExceptionHandlingExample.rar)

That’s it friends, hope you enjoy the article 😉

​ ​​