**RESTful Web Services (JAX-RS) Tutorials**

|  |  |  |
| --- | --- | --- |
| [Web Services](http://www.java4s.com/web-services/) » On Jul 5, 2014 By [Sivateja](https://plus.google.com/118054670710951892925?rel=author" \t "_blank) |  | [Tweet](http://twitter.com/share) |

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 ;)

## JAX-RS Tutorials

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

# What is Web Services, Web Services Introduction

|  |  |  |
| --- | --- | --- |
| [Web Services](http://www.java4s.com/web-services/) » On Jul 6, 2014 By [Sivateja](https://plus.google.com/118054670710951892925?rel=author" \t "_blank) |  | [Tweet](http://twitter.com/share) |

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,

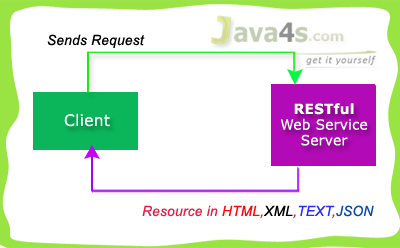
**SOAP** [Simple Object Access Protocol] and

**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, CXFfrom 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

|  |  |  |
| --- | --- | --- |
| [Web Services](http://www.java4s.com/web-services/) » On Jul 6, 2014 By [Sivateja](https://plus.google.com/118054670710951892925?rel=author" \t "_blank) |  | [Tweet](http://twitter.com/share) |

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, @FormParamannotations :-) , i will talk more about these annotations in the next examples.

# How RESTful Web Services Extract Input Parameters

|  |  |  |
| --- | --- | --- |
| [Web Services](http://www.java4s.com/web-services/) » On Jul 6, 2014 By [Sivateja](https://plus.google.com/118054670710951892925?rel=author" \t "_blank) |  | [Tweet](http://twitter.com/share) |

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](http://www.java4s.com/web-services/) » On Jul 6, 2014 By [Sivateja](https://plus.google.com/118054670710951892925?rel=author" \t "_blank) |  | [Tweet](http://twitter.com/share) |

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 thisarticle for help [ [How to Install m2eclipse (Maven) Plugin in Eclipse](http://www.java4s.com/core-java/how-to-install-m2eclipse-maven-plugin-in-eclipse/) ]

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

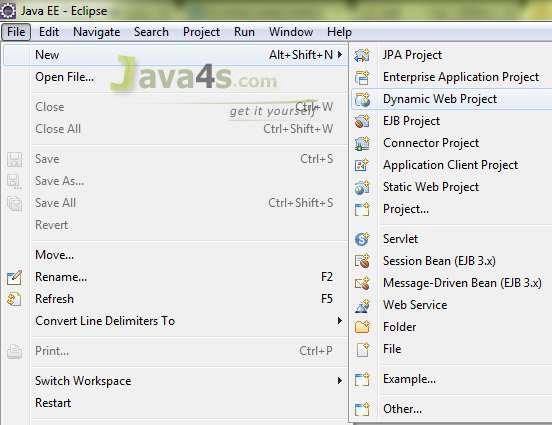
## Required

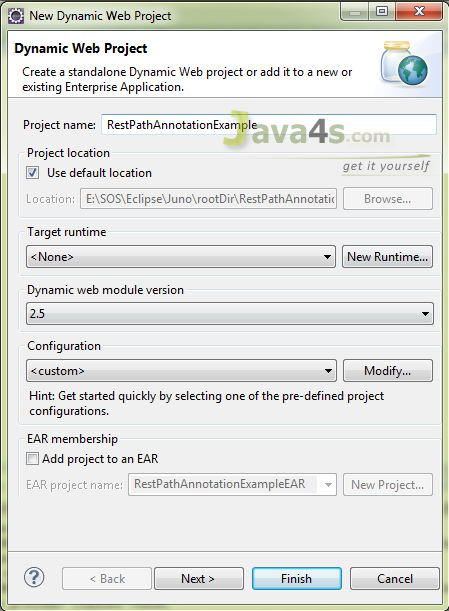
* Eclipse Juno
* JDK 1.6
* Jersey 1.8
* Maven Plugin
* Tomcat 6.0 Server

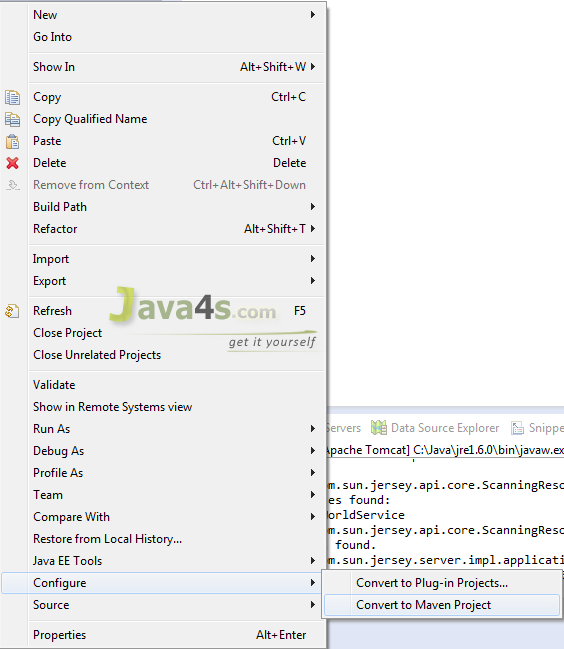
## Steps

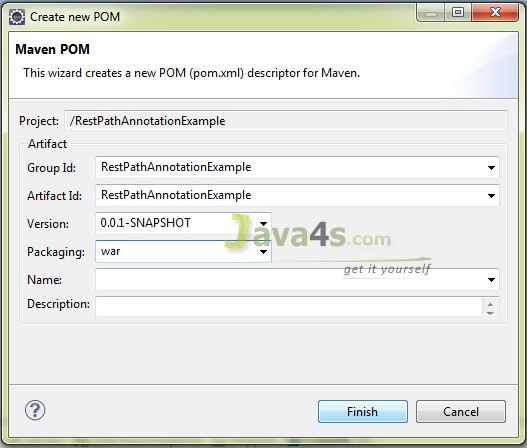
* Create a ‘Dynamic Web Project’
* Convert the project into ‘Maven Project’ [ Of course you can also create Maven project directly ]
* Add required dependencies in ‘pom.xml‘
* Change web.xml [ register com.sun.jersey.spi.container.servlet.ServletContainer and add related init-param]
* Run the application

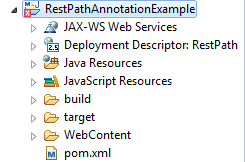
## Steps to Create Restful Web Services in Eclipse

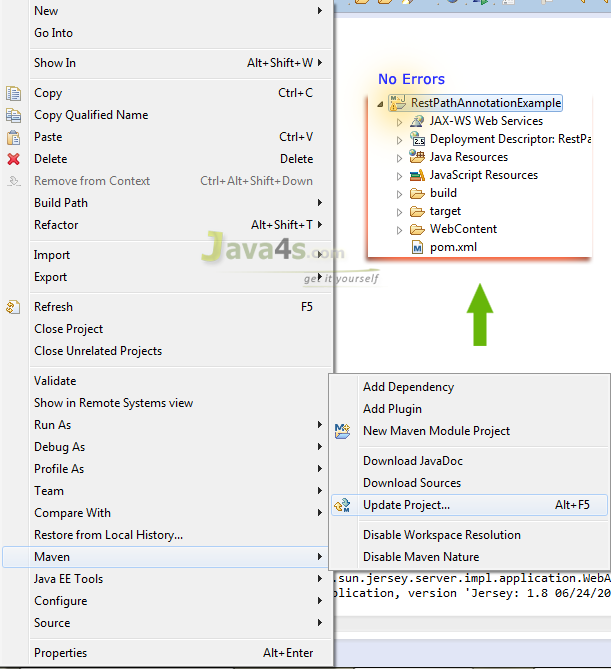
Open Eclipse > File > New > Dynamic Web Project  


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

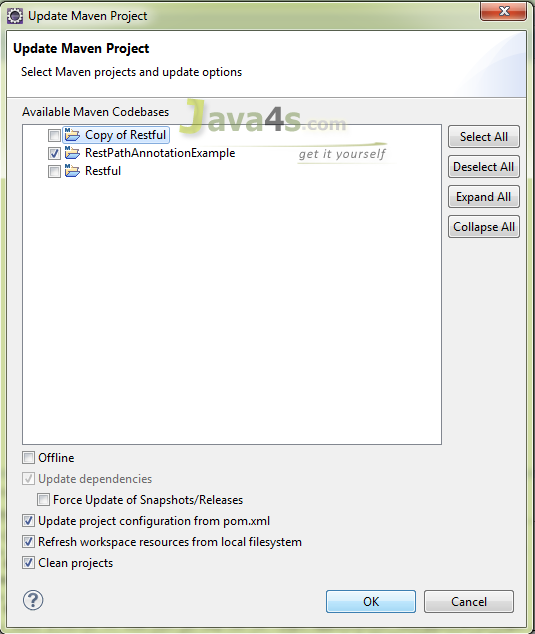
## Pom.xml

**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.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | <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



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | 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



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <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…

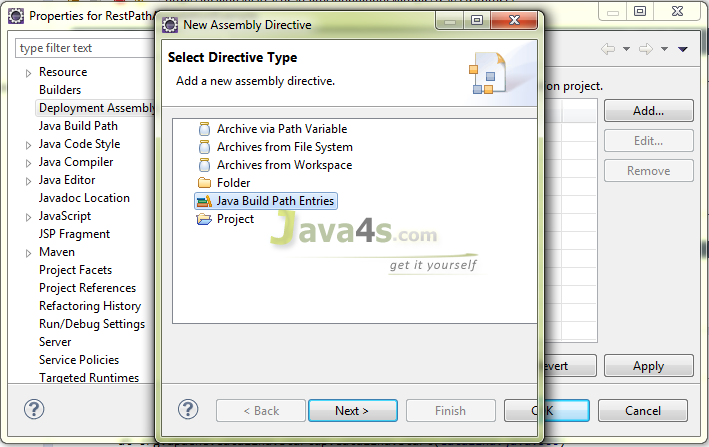
Main application URL:  
http://localhost:2013/RestPathAnnotationExample/

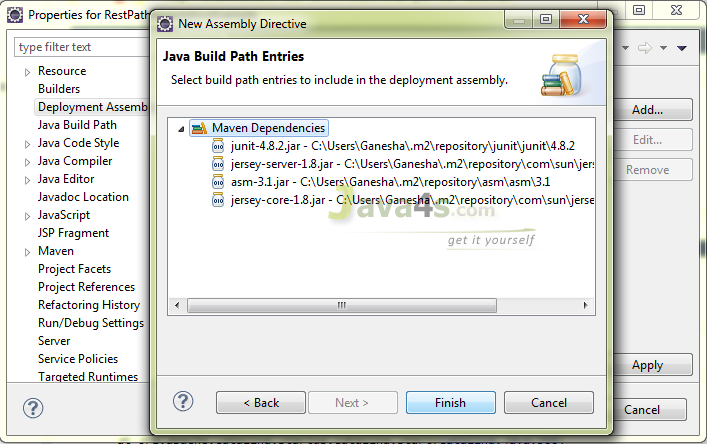
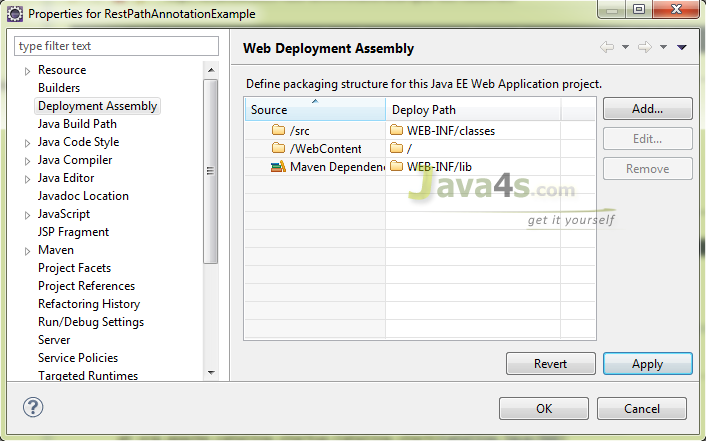
Web.xml URL pattern:  
/rest

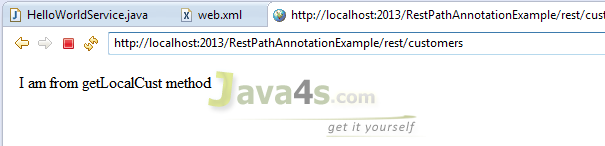
@Path in HelloWorldService.java:  
/customers

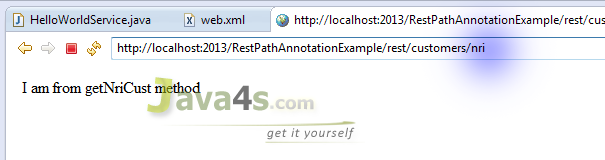
Final URL should be  
**http://localhost:2013/RestPathAnnotationExample/rest/customers**

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](http://www.java4s.com/web-services/) » On Jul 8, 2014 By [Sivateja](https://plus.google.com/118054670710951892925?rel=author" \t "_blank) |  | [Tweet](http://twitter.com/share) |

In RESTful (JAX-RS) web services @PathParam annotation will be used to bind RESTfulURL 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 completestepbystepflow of JAX-RS, Go through this article ‘[*Jersey Hello World example With Maven in Eclipse Juno*](http://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



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | <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



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | <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

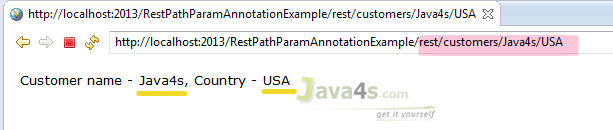


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | package com.java4s;    import javax.ws.rs.GET;  import javax.ws.rs.Path;  import javax.ws.rs.PathParam;  import javax.ws.rs.Produces;  import javax.ws.rs.core.Response;    @Path("/customers")  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 inRestServicePathParamJava4s.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 behttp://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 Stringname,Stringcountry respectively.  
  
* Check the output

## Output



[[http://www.java4s.com/wp-content/uploads/2011/05/download.png](http://www.java4s.com/wp-content/web-services/Rest-Path-Param-Annotation-Example.rar)](http://www.java4s.com/wp-content/web-services/Rest-Path-Param-Annotation-Example.rar)

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

|  |  |  |
| --- | --- | --- |
| [Web Services](http://www.java4s.com/web-services/) » On Jul 8, 2014 By [Sivateja](https://plus.google.com/118054670710951892925?rel=author" \t "_blank) |  | [Tweet](http://twitter.com/share) |

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 theparameters 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, countryKeyand their values are Java4s, USArespectively, 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

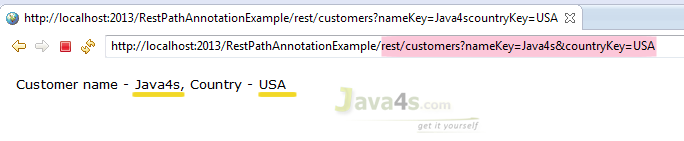


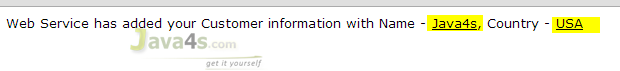
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package com.java4s;    import javax.ws.rs.GET;  import javax.ws.rs.Path;  import javax.ws.rs.Produces;  import javax.ws.rs.QueryParam;  import javax.ws.rs.core.Response;    @Path("/customers")  public 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 inRestServiceQueryParamJava4s.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 thehttp://localhost:2013/RestQueryParamAnnotationExample/rest/customers?nameKey=java4s&countryKey=USA
* Check the output by hitting the above URL

## Output



Output:  


[[http://www.java4s.com/wp-content/uploads/2011/05/download.png](http://www.java4s.com/wp-content/web-services/Rest-Form-Param-Annotation-Example.rar)](http://www.java4s.com/wp-content/web-services/Rest-Form-Param-Annotation-Example.rar)

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

|  |  |  |
| --- | --- | --- |
| [Web Services](http://www.java4s.com/web-services/) » On Jul 9, 2014 By [Sivateja](https://plus.google.com/118054670710951892925?rel=author" \t "_blank) |  | [Tweet](http://twitter.com/share) |

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*](http://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](http://www.java4s.com/web-services/jersey-hello-world-example-using-jax-rs-specification/)
* RestServiceFormParamJava4s.java
* Client.html

## RestServiceFormParamJava4s.java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | package 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



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | <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> |

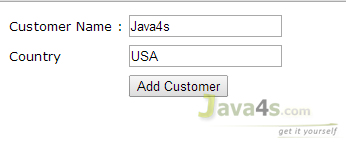
## Explanation

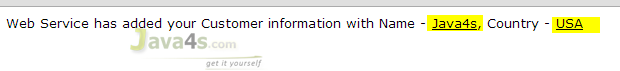
* Right click on your project root folder >Run As >Run on Server
* Eclipse will open http://localhost:2013/<projectRootFolder> with 404 Error by default, forget about that
* Now open Client.htmlin your web browser, enter the details and click submit [ I have created this .html file to send input form 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](http://www.java4s.com/web-services/) » On Jul 10, 2014 By [Sivateja](https://plus.google.com/118054670710951892925?rel=author" \t "_blank) |  | [Tweet](http://twitter.com/share) |

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](http://www.java4s.com/web-services/jersey-hello-world-example-using-jax-rs-specification/), i am using the same xml’s ]
* RestServiceFileDownloadJava4s.java

## RestServiceFileDownloadJava4s.java



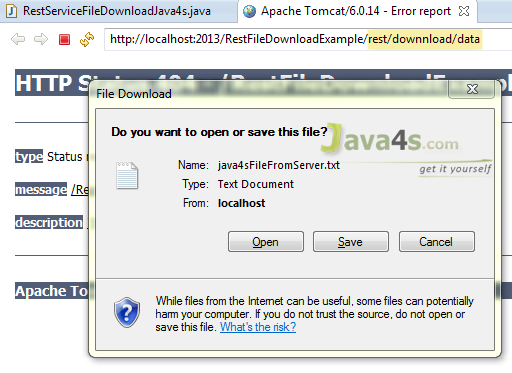
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | package com.java4s;    import java.io.File;import javax.ws.rs.GET;  import javax.ws.rs.Path;  import javax.ws.rs.Produces;  import javax.ws.rs.core.Response;  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 annotateour 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



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

|  |  |  |
| --- | --- | --- |
| [Web Services](http://www.java4s.com/web-services/) » On Jul 20, 2014 By [Sivateja](https://plus.google.com/118054670710951892925?rel=author" \t "_blank) |  | [Tweet](http://twitter.com/share) |

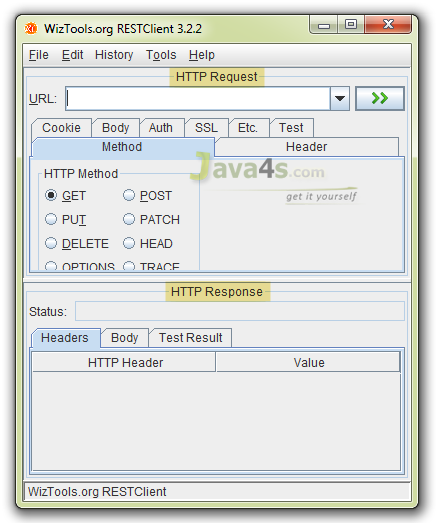
In this article i will show you how to test RESTful web service (JAX-RS), so far we havelearned how to create a RESTful service and testing GET and POST requests through some web browser.  But in real time projects we will use different tools to test RESTful web services.  If you would like to test JAX-RS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



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | <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



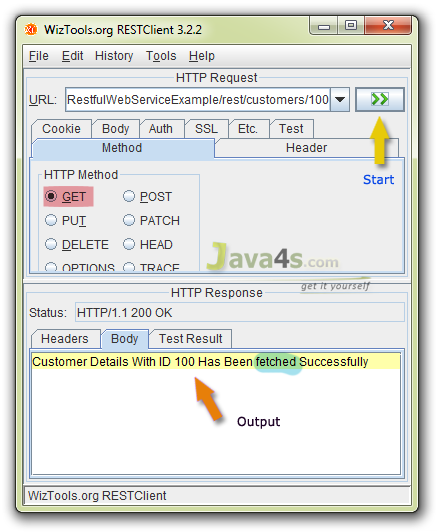
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | <web-app id="WebApp\_ID" version="2.4"      xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee      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

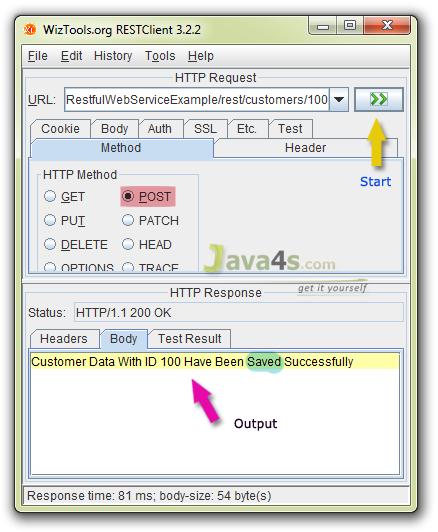


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | package com.java4s;    import javax.ws.rs.DELETE;  import javax.ws.rs.GET;  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();       }    } |

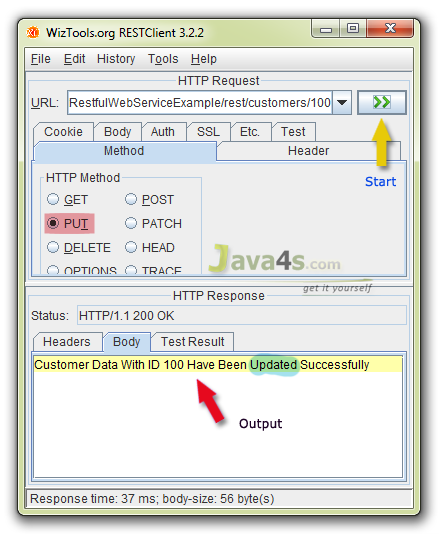
## Testing JAX-RS GET Request

* 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**  
  

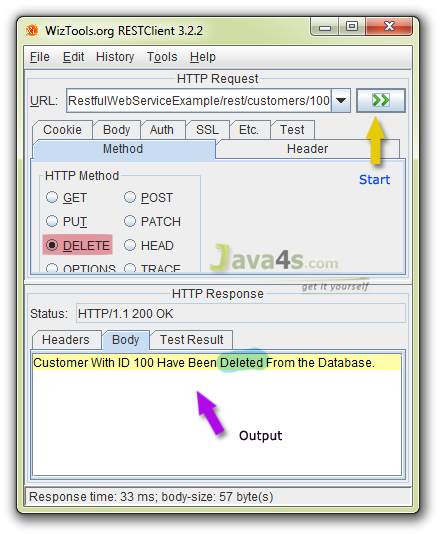
## Testing JAX-RS POST Request

* 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  
  

[[http://www.java4s.com/wp-content/uploads/2011/05/download.png](http://www.java4s.com/wp-content/web-services/Test-Restful-WebService-Example.rar)](http://www.java4s.com/wp-content/web-services/Test-Restful-WebService-Example.rar)