**REST Web Service**

REST stands for Representational State Transfer. REST was a term coined by Roy Fielding in his doctoral dissertation. It is an architecture style for for creating network based applications.  Key properties of REST are client-server communication, stateless protocol, cacheable, layered implementation and uniform interface. REST is sometimes seen as an alternate for SOAP. Refer the earlier written tutorial to understand the [difference between REST and SOAP](https://javapapers.com/web-service/rest-vs-soap/).

In REST architecture resources are accessed using an interface over HTTP or similar protocols. Mostly HTTP is used over methods GET, PUT, POST, DELETE wherein we can use other protocols also in REST architecture. Resources are uniquely identified using URI. Web services that conforms to the constraints of REST are called RESTful web services. To know about web service in general refer the [web service introduction tutorial](https://javapapers.com/web-service/web-service-introduction-tutorial/).

**Java JAX-RS and Jersey**

JAX-RS Specification is the Java API for RESTful web services. JAX-RS specification is the outcome of the [Java Specification Request](https://javapapers.com/core-java/java-community-process-jcp-and-java-specification-request-jsr/) (JSR) 311, 339. JAX-RS uses the declarative style of programming using annotations.  JAX-RS provides high level simpler API to write RESTful web services that can run on Java EE and SE platforms.

Jersey is the open source reference implementation of Java JAX-RS specification. It provides a Java library using which we can easily create RESTful web services in Java platform. JAX-RS / Jersey supports JAXB based XML bindings. JAXB provides API to access and process XML documents, to know more refer [JAXB tutorial](https://javapapers.com/jee/jaxb-tutorial/).

Download the Jersey distribution bundle from [Jersey download page](https://jersey.java.net/download.html).

**RESTful Web Service Example**

[Download](https://javapapers.com/wp-content/uploads/2015/02/RESTful-Jersey-Hello-World.zip)

Let us now dive into the example RESTful web services project. It is a simple example, we will create a RESTful web service with a resource that will respond with hello world text.

I have used Eclipse WTP and Tomcat container. If you have an Eclipse with JEE perspective, then you are good to go.

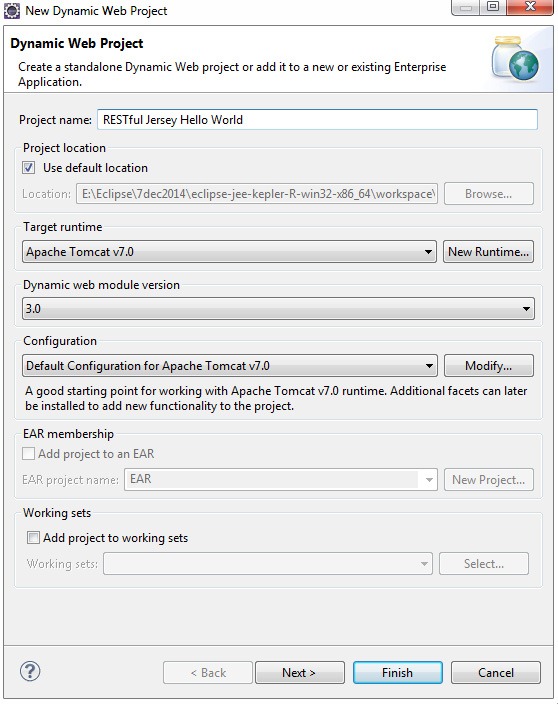
JAXRS – Android - Spec

RESTEASY

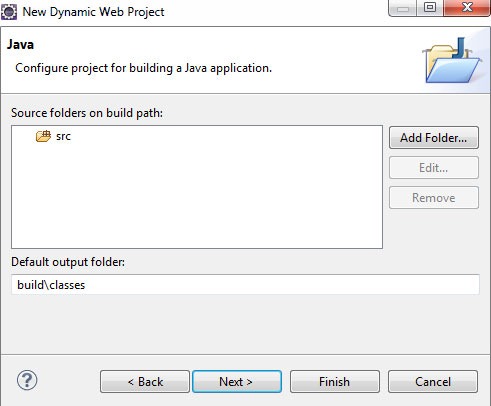
Jersey

**1. Create New Dynamic Web Project**

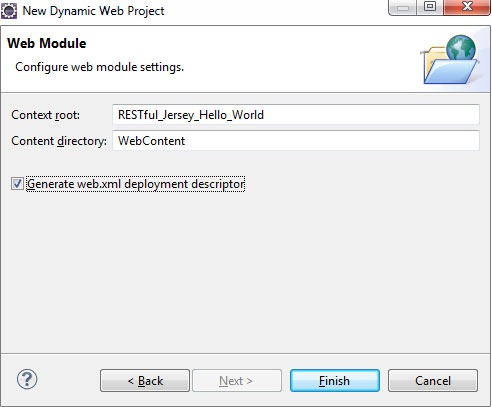
Create a new dynamic web project using the Eclipse WTP wizard.

[](https://javapapers.com/wp-content/uploads/2015/02/NewWebServiceProject.jpg)

Just Click Next.

[](https://javapapers.com/wp-content/uploads/2015/02/NewRESTfulWebServiceProject.jpg)

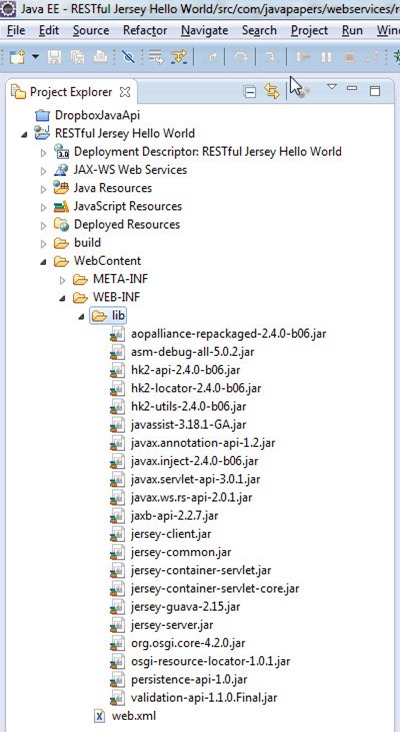
Just Click Next.

[](https://javapapers.com/wp-content/uploads/2015/02/WebServiceProjectXMLDescriptor.jpg)

Enable the “Generate web.xml deployment descriptor” checkbox so that Eclipse will generate a web.xml.

**2. Add JAX-RS / Jersey Dependent JAR files**

Download the Jersey bundle from its website. It has a .zip file and it contains javax.ws.rs-api.jar then its dependencies and external dependencies. Remember to add all those three set of JARs to the lib folder in project as shown below.

[](https://javapapers.com/wp-content/uploads/2015/02/JerseyRESTfulDependencies.jpg)

**3. HelloWorld RESTful Web Service Resource**

Create the resource file as shown below in the Java sources.

package com.javapapers.webservices.rest.jersey;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

@Path("/helloworld")

public class HelloWorld {

@GET

@Produces(MediaType.TEXT\_PLAIN)

public String sayPlainTextHello() {

return "Hello World RESTful Jersey!";

}

@GET

@Produces(MediaType.TEXT\_XML)

public String sayXMLHello() {

return "<?xml version=\"1.0\"?>" + "<hello> Hello World RESTful Jersey"

+ "</hello>";

}

@GET

@Produces(MediaType.TEXT\_HTML)

public String sayHtmlHello() {

return "<html> " + "<title>" + "Hello World RESTful Jersey"

+ "</title>" + "<body><h1>" + "Hello World RESTful Jersey"

+ "</body></h1>" + "</html> ";

}

}

**4. web.xml Servlet Mappings**

[Servlet mapping](https://javapapers.com/servlet/what-is-servlet-mapping/) should be updated in the web.xml to point to our web service resource.

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

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" id="WebApp\_ID" version="3.0">

<display-name>RESTful Jersey Hello World</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>RESTful Jersey Hello World Service</servlet-name>

<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>jersey.config.server.provider.packages</param-name>

<param-value>com.javapapers.webservices.rest.jersey</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>RESTful Jersey Hello World Service</servlet-name>

<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

</web-app>

**5. Run the RESTful Web Service**

“Run on Server” the web service application. The RESTful web service resource we created can be accessed from a browser as below,

[](https://javapapers.com/wp-content/uploads/2015/02/JerseyRESTfulHelloWorldService.jpg)

**6. RESTful Web Service Client**

As a add-on, lets have a look at how to consume this RESTful webservice using a Java Jersey client. Jersey provides  a RESTful client library and using it we can consume the above web service.

package com.javapapers.webservices.rest.jersey;

import java.net.URI;

import javax.ws.rs.client.Client;

import javax.ws.rs.client.ClientBuilder;

import javax.ws.rs.client.WebTarget;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import javax.ws.rs.core.UriBuilder;

import org.glassfish.jersey.client.ClientConfig;

public class RESTfulJerseyClient {

private static final String webServiceURI = "http://localhost:8080/RESTful\_Jersey\_Hello\_World";

public static void main(String[] args) {

ClientConfig clientConfig = new ClientConfig();

Client client = ClientBuilder.newClient(clientConfig);

URI serviceURI = UriBuilder.fromUri(webServiceURI).build();

WebTarget webTarget = client.target(serviceURI);

// response

System.out.println(webTarget.path("rest").path("helloworld").request()

.accept(MediaType.TEXT\_PLAIN).get(Response.class).toString());

// text

System.out.println(webTarget.path("rest").path("helloworld").request()

.accept(MediaType.TEXT\_PLAIN).get(String.class));

// xml

System.out.println(webTarget.path("rest").path("helloworld").request()

.accept(MediaType.TEXT\_XML).get(String.class));

// html

System.out.println(webTarget.path("rest").path("helloworld").request()

.accept(MediaType.TEXT\_HTML).get(String.class));

}

}

**Output of the RESTful Service Client**

InboundJaxrsResponse{ClientResponse{method=GET, uri=http://localhost:8080/RESTful\_Jersey\_Hello\_World/rest/helloworld, status=200, reason=OK}}

Hello World RESTful Jersey!

<?xml version="1.0"?><hello> Hello World RESTful Jersey</hello>

<html> <title>Hello World RESTful Jersey</title><body><h1>Hello World RESTful Jersey</body></h1></html>