<http://theopentutorials.com/examples/java-ee/jax-rs/create-a-simple-restful-web-service-using-jersey-jax-rs/>

# **HOW TO CREATE A SIMPLE RESTFUL WEB SERVICE USING JERSEY JAX RS API**

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

RESTful web services are based on HTTP protocol and its methods PUT, GET, POST, and DELETE. These web services are better integrated with HTTP than SOAP-based services are, and as such do not require XML SOAP messages or WSDL service definitions.

Java provides support for Restful web services through Java API for RESTful Web Services JAX-RS. The complete specification is available as JSR 311. JAX-RS is an annotation-based API for implementing RESTful web services, based on HTTP, in Java. Essentially, classes and methods are annotated with information that enables a runtime to expose them as resources – an approach that is very different from the one exposed by the servlet programming model. A runtime that implements JAX-RS mediates between the HTTP protocol and the Java classes, taking into account URIs, requested and accepted content types, and HTTP methods. **Jersey is a production quality reference implementation of JAX-RS API**.

**Contents**[[hide](http://theopentutorials.com/examples/java-ee/jax-rs/create-a-simple-restful-web-service-using-jersey-jax-rs/)]

* [1 Project Description](http://theopentutorials.com/examples/java-ee/jax-rs/create-a-simple-restful-web-service-using-jersey-jax-rs/#Project_Description)
* [2 Environment Used](http://theopentutorials.com/examples/java-ee/jax-rs/create-a-simple-restful-web-service-using-jersey-jax-rs/#Environment_Used)
* [3 Creating and Publishing Restful Web Service](http://theopentutorials.com/examples/java-ee/jax-rs/create-a-simple-restful-web-service-using-jersey-jax-rs/#Creating_and_Publishing_Restful_Web_Service)
* [4 Creating a Restful Web Service Client](http://theopentutorials.com/examples/java-ee/jax-rs/create-a-simple-restful-web-service-using-jersey-jax-rs/#Creating_a_Restful_Web_Service_Client)
* [5 Output](http://theopentutorials.com/examples/java-ee/jax-rs/create-a-simple-restful-web-service-using-jersey-jax-rs/#Output)
* [6 Project Structure](http://theopentutorials.com/examples/java-ee/jax-rs/create-a-simple-restful-web-service-using-jersey-jax-rs/#Project_Structure)

Project Description

* In this example, we create a simple Java Calculator class with operations ‘add’ and ‘subtract’.
* We expose these as restful web services which accepts HTTP GET Requests and sends the response as XML or Plain Text.
* The restful web service is made available at http://localhost:9999/calcrest/calc/add/ and http://localhost:9999/calcrest/calc/sub/
* We send the parameters in the URI itself. For example, to add 20 and 30, the URI is, http://localhost:9999/calcrest/calc/add/20/30
* We then create a web service client which sends a HTTP GET Request to the mentioned URI and the response (result of the invoked web service) is displayed.

Environment Used

* Java SE 6
* Jersey (JAX RS 1.1)
* Download the zip of Jersey files from this location –<http://jersey.java.net/nonav/documentation/latest/chapter_deps.html>
* Eclipse IDE

Creating and Publishing Restful Web Service

1. Create a Java Project ‘CalcJAXRS’.
2. In the Project Build Path add the following jars from the downloaded Jersey zip file.
   * Jersey core
   * Jersey server
   * Jersey client
   * JSR 311 API
   * asm 3.x
3. Create a package ‘com.theopentutorials.jaxrs.calc’
4. Create a Java class ‘CalcREST’ and type the following code.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  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 | package com.theopentutorials.jaxrs.calc;    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("/calc")  public class CalcREST {        @GET      @Path("/add/{a}/{b}")      @Produces(MediaType.TEXT\_PLAIN)      public String addPlainText(@PathParam("a") double a, @PathParam("b") double b) {          return (a + b) + "";      }    @GET      @Path("/sub/{a}/{b}")      @Produces(MediaType.TEXT\_PLAIN)      public String subPlainText(@PathParam("a") double a, @PathParam("b") double b) {          return (a - b) + "";      }        @GET      @Path("/add/{a}/{b}")      @Produces(MediaType.TEXT\_XML)      public String add(@PathParam("a") double a, @PathParam("b") double b) {          return "<?xml version=\"1.0\"?>" + "<result>" +  (a + b) + "</result>";      }        @GET      @Path("/sub/{a}/{b}")      @Produces(MediaType.TEXT\_XML)      public String sub(@PathParam("a") double a, @PathParam("b") double b) {          return "<?xml version=\"1.0\"?>" + "<result>" +  (a - b) + "</result>";      }  } |

1. **@Path** may be used on classes and such classes are referred to as root resource classes. @PATH(“calc”) sets the path to the base URI + /calc. The base URI consists of the host, port and any context. We will set this base URI on creating the server.
2. @Path may also be used on methods of root resource classes. This enables common functionality for a number of resources to be grouped together and potentially reused. So for example @PATH(“/add”) on the method signature indicates the URI path for this method to be invoked is base URI + /calc/add.
3. To retrieve the parameters from the URI and get those values as variables, we use URI Path Templates which are variables enclosed within curly braces “{“ and “}” and are replaced at runtime to respond to a request based on the substituted URI. To obtain the value of the variable the @PathParam may be used on method parameter of a request method.
   * For example, for the relative URI /calc/add/20/30 with @Path(“/add/{a}/{b}”) and method signature String add(@PathParam(“a”) double a, @PathParam(“b”), the values 20 and 30 will be assigned to the variables a and b respectively.
4. **@GET** annotation is used to indicate that the annotated method will be invoked in response to a HTTP GET request.
5. **@Produces** annotation is used to specify the MIME media types of representations a resource can produce and send back to the client. In our example, our web service will produce representations identified by the MIME media type “text/plain” and “text/xml” which are specified using the static fields TEXT\_PLAIN and TEXT\_XML of MediaType class.
6. Now we need to start a HTTP server to publish our web service and accept requests.
7. Create a class “CalcRESTStartUp” and type the following code.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | package com.theopentutorials.jaxrs.calc;    import java.io.IOException;  import com.sun.jersey.api.container.httpserver.HttpServerFactory;  import com.sun.net.httpserver.HttpServer;    public class CalcRESTStartUp {        static final String BASE\_URI = "<http://localhost:9999/calcrest/>";        public static void main(String[] args) {          try {              HttpServer server = HttpServerFactory.create(BASE\_URI);              server.start();              System.out.println("Press Enter to stop the server. ");              System.in.read();              server.stop(0);          } catch (IllegalArgumentException e) {              e.printStackTrace();          } catch (IOException e) {              e.printStackTrace();          }      }  } |

1. We use the Jersey’s HttpServerFactory to create a server at the mentioned BASE URI. Make sure that port number 9999 is not used by any other application. If so, use any other unused port.
2. Run this class “CalcRESTStartup” as Java Application.
3. Our class “CalcREST” should be automatically identified as root resource class and should be deployed.
4. To test it, open a browser and type the following URL

http://localhost:9999/calcrest/application.wadl

. A Web Application Description Language (WADL) should be displayed. This is similar to WSDL. WADL is not standardized by W3C.

1. To test the service we can type the following URLs in the browser

http://localhost:9999/calcrest/calc/add/20/30/

and

http://localhost:9999/calcrest/calc/sub/20/30/

1. The output on the browser will be in XML format even though we have our service produce plain text. This is because of the accept type HTTP header sent by the browser. By default the browser sends the text/html accept type and other xhtml and xml types. Since our service does not accept html and seeing that the browser accepts XML, it produces the output in XML.

Creating a Restful Web Service Client

Even though the web service is accessible via browser, the primary purpose of web service is for machines to communicate. So let us access this web service programmatically by creating a client.

1. Create a new package “com.theopentutorials.jaxrs.calc.client”.
2. Create a new class “CalcRESTClient” in that package and type the following code.

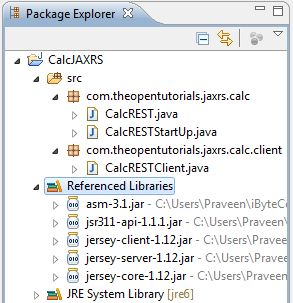
|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | package com.theopentutorials.jaxrs.calc.client;    import javax.ws.rs.core.MediaType;  import com.sun.jersey.api.client.Client;  import com.sun.jersey.api.client.ClientResponse;  import com.sun.jersey.api.client.WebResource;  import com.sun.jersey.api.client.config.ClientConfig;  import com.sun.jersey.api.client.config.DefaultClientConfig;    public class CalcRESTClient {      static final String REST\_URI = "<http://localhost:9999/calcrest>";      static final String ADD\_PATH = "calc/add";      static final String SUB\_PATH = "calc/sub";        public static void main(String[] args) {            int a = 122;          int b = 34;            ClientConfig config = new DefaultClientConfig();          Client client = Client.create(config);          WebResource service = client.resource(REST\_URI);            WebResource addService = service.path(ADD\_PATH).path(a + "/" + b);          System.out.println("Add Response: " + getResponse(addService));          System.out.println("Add Output as XML: " + getOutputAsXML(addService));          System.out.println("Add Output as Text: " + getOutputAsText(addService));          System.out.println("---------------------------------------------------");            WebResource subService = service.path(SUB\_PATH).path(a + "/" + b);          System.out.println("Sub Response: " + getResponse(subService));          System.out.println("Sub Output as XML: " + getOutputAsXML(subService));          System.out.println("---------------------------------------------------");        }        private static String getResponse(WebResource service) {          return service.accept(MediaType.TEXT\_XML).get(ClientResponse.class).toString();      }        private static String getOutputAsXML(WebResource service) {          return service.accept(MediaType.TEXT\_XML).get(String.class);      }        private static String getOutputAsText(WebResource service) {          return service.accept(MediaType.TEXT\_PLAIN).get(String.class);      }  } |

1. In the client, we create a web resource object based on the provided URI and a default client config.
2. We specify the “add” and “sub” resource paths and include parameters “a” and “b” to it.
3. We also need to specify the accept type of the client request and HTTP method as GET by using the methods “accept” and “get”. The “get” method parameter specifies the format to convert the output.
4. Run this client application.

Output

Add Response: GET http://localhost:9999/calcrest/calc/add/122/34 returned a response status of 200 OK  
Add Output as XML: 156.0  
Add Output as Text: 156.0  
—————————————————  
Sub Response: GET http://localhost:9999/calcrest/calc/sub/122/34 returned a response status of 200 OK  
Sub Output as XML: 88.0  
—————————————————

Project Structure

[](http://theopentutorials.com/totwp331/wp-content/uploads/how-to-create-a-simple-restful-web-service-using-jersey-jax-rs-api_2542/jax-rs-pjt-structure.jpg?c3a9c1)