# How to read multiple values of a query paramter in JAX-RS restful web services?

In the previous example we have seen how to read query paramters from the request using @QueryParam annotation. What if the user request is in the below format? same id with multiple values?

**http://localhost:8080/RestfulWebServices/employee/query?id=1016&id=1017&id=1415**

In the above URL, the id value is same, but with different values, this can be easily readable as a list of values using UriInfo object.

In the previous examples we have given details of application setup, dependencies, web.xml file configurations: If you want to know about these configuration, please refer these:

|  |  |
| --- | --- |
| |  | | --- | | package com.javacofee.restful;    import java.util.List;    import javax.ws.rs.GET;  import javax.ws.rs.Path;  import javax.ws.rs.core.Context;  import javax.ws.rs.core.Response;  import javax.ws.rs.core.UriInfo;    @Path("/employee")  public class QueryParamExampleService {        @GET      @Path("/query")      public Response getEmployeeQuery(@Context UriInfo uriInfo){            /\*\* read complete employee id list from request query parameter\*\*/          List<String> empIdList = uriInfo.getQueryParameters().get("id");          System.out.println("Received List: "+empIdList);          /\*\* read first employee id from request query parameter \*\*/          String firstEmpId = uriInfo.getQueryParameters().getFirst("id");          System.out.println("First emp record from the request: "+firstEmpId);            return Response.status(200).entity("processed request").build();      }  } | |

In the above example, if you use "**/employee/query?id=1016&id=1017&id=1415**" URI pattern with query parameters, here is the output:

|  |
| --- |
| Output: |
| INFO [stdout] (http--127.0.0.1-8080-1) Received List: [1016, 1017, 1415]  INFO [stdout] (http--127.0.0.1-8080-1) First emp record from the request: 1016 |

# How to pass header parameters as method inputs in JAX-RS restful web services?

In this page you can see an example to pass http header info as a method input using @HeaderParam annotation.

In the previous examples we have given details of application setup, dependencies, web.xml file configurations: If you want to know about these configuration, please refer these:

|  |  |
| --- | --- |
| |  | | --- | | package com.javacoffee.restful;    import javax.ws.rs.GET;  import javax.ws.rs.HeaderParam;  import javax.ws.rs.Path;  import javax.ws.rs.core.Response;    @Path("/http-header")  public class HttpHeaderService {        @GET      @Path("query")      public Response queryHeaderInfo(@HeaderParam("Cache-Control") String ccControl,                                          @HeaderParam("User-Agent") String uaStr){            String resp = "Received http headers are Cache-Control: "+ccControl+                          "<br>User-Agent: "+uaStr;          return Response.status(200).entity(resp).build();      }  } | |

In the above example, if you use "**http://localhost:8080/RestfulWebServices/http-header/query**" URI pattern, here is the output:

|  |
| --- |
| Output: |
| Received http headers are Cache-Control: max-age=0  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_9\_2) AppleWebKit/537.36 (KHTML, like Gecko)  Chrome/38.0.2125.104 Safari/537.36 |

# How to read header parameters in JAX-RS restful web services?

In this page you can see an example to query http header info using @Context annotation and HttpHeaders.

In the previous examples we have given details of application setup, dependencies, web.xml file configurations: If you want to know about these configuration, please refer these:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | |  | | --- | | package com.javacoffee.restful;    import java.util.Set;    import javax.ws.rs.GET;  import javax.ws.rs.Path;  import javax.ws.rs.core.Context;  import javax.ws.rs.core.HttpHeaders;  import javax.ws.rs.core.Response;    @Path("/http-header")  public class HttpHeaderService {        @GET      @Path("query")      public Response queryHeaderInfo(@Context HttpHeaders httpHeaders){            /\*\* how to get specific header info? \*\*/          String cacheControl = httpHeaders.getRequestHeader("Cache-Control").get(0);          System.out.println("Cache-Control: "+cacheControl);          /\*\* get list of all header parameters from request \*\*/          Set<String> headerKeys = httpHeaders.getRequestHeaders().keySet();          for(String header:headerKeys){              System.out.println(header+":"+httpHeaders.getRequestHeader(header).get(0));          }          return Response.status(200).entity("successfully queried header info").build();      }  } | |   Test with "**http://localhost:8080/RestfulWebServices/http-header/query**" URL, here is the output: |
| |  | | --- | | Output: | | Cache-Control: max-age=0  [stdout] (http--127.0.0.1-8080-1) host:localhost:8080  [stdout] (http--127.0.0.1-8080-1) connection:keep-alive  [stdout] (http--127.0.0.1-8080-1) cache-control:max-age=0  [stdout] (http--127.0.0.1-8080-1) accept:text/html,application/xhtml+xml,application/xml;q=0.9,  image/webp,\*/\*;q=0.8  [stdout] (http--127.0.0.1-8080-1) user-agent:Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_9\_2) AppleWebKit/537.36  (KHTML, like Gecko) Chrome/38.0.2125.104 Safari/537.36  [stdout] (http--127.0.0.1-8080-1) accept-encoding:gzip,deflate,sdch  [stdout] (http--127.0.0.1-8080-1) accept-language:en-US,en;q=0.8  [stdout] (http--127.0.0.1-8080-1) cookie:\_\_atuvc=24%7C23%2C0%7C24%2C3%7C25%2C45%7C26%2C41%7C27;  bsau=14136427273262128776 | |

# How to upload file using Jersey restful web services?

In this page you will see an example for how to upload a file using Jersey API in restful web services.

In order to implement file upload feature, include jersey-multipart dependency in your pom.xml file:

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/upload-file-jersey/)   |  |  | | --- | --- | | 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 | <?xml version="1.0" encoding="UTF-8"?>  <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>RestfulWebServices</groupId>      <artifactId>RestfulWebServices</artifactId>      <version>0.0.1-SNAPSHOT</version>      <packaging>war</packaging>      <dependencies>          <dependency>              <groupId>com.sun.jersey</groupId>              <artifactId>jersey-server</artifactId>              <version>1.17</version>          </dependency>          <dependency>              <groupId>com.sun.jersey</groupId>              <artifactId>jersey-servlet</artifactId>              <version>1.17</version>          </dependency>          <dependency>              <groupId>com.sun.jersey.contribs</groupId>              <artifactId>jersey-multipart</artifactId>              <version>1.17</version>          </dependency>      </dependencies>  </project> | |

web.xml file reference:

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/upload-file-jersey/)   |  |  | | --- | --- | | 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">        <servlet>          <servlet-name>jersey-serlvet</servlet-name>              <servlet-class>                       com.sun.jersey.spi.container.servlet.ServletContainer                  </servlet-class>          <init-param>               <param-name>jersey.config.server.provider.packages</param-name>               <param-value>com.java2novice.restful</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> | |

Here is the HTML upload form:

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/upload-file-jersey/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"                  "<http://www.w3.org/TR/html4/loose.dtd>">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Upload File Example</title>  </head>  <body>      <h1>Upload File</h1>      <form action="rest/test/upload" method="post" enctype="multipart/form-data">          <p>Select a file : <input type="file" name="file"/></p>          <input type="submit" value="Upload File" />      </form>  </body>  </html> | |

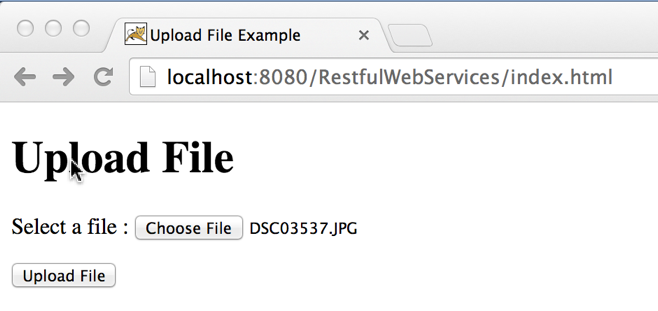
Here is the upload service class, we are using @FormDataParam annotation to receive uploaded file and FormDataContentDisposition to receive file properties like name, header, etc.

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/upload-file-jersey/)   |  |  | | --- | --- | | 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 | package com.javacoffee.restful;    import java.io.File;  import java.io.FileOutputStream;  import java.io.IOException;  import java.io.InputStream;  import java.io.OutputStream;    import javax.ws.rs.Consumes;  import javax.ws.rs.POST;  import javax.ws.rs.Path;  import javax.ws.rs.Produces;  import javax.ws.rs.core.MediaType;  import com.sun.jersey.core.header.FormDataContentDisposition;  import com.sun.jersey.multipart.FormDataParam;    @Path("test")  public class RestUploadService {        private static final String FOLDER\_PATH = "C:\my\_files\";        @POST      @Path("/upload")      @Consumes(MediaType.MULTIPART\_FORM\_DATA)      @Produces(MediaType.TEXT\_PLAIN)      public String uploadFile(@FormDataParam("file") InputStream fis,                      @FormDataParam("file") FormDataContentDisposition fdcd) {            OutputStream outpuStream = null;          String fileName = fdcd.getFileName();          System.out.println("File Name: " + fdcd.getFileName());          String filePath = FOLDER\_PATH + fileName;            try {              int read = 0;              byte[] bytes = new byte[1024];              outpuStream = new FileOutputStream(new File(filePath));              while ((read = fis.read(bytes)) != -1) {                  outpuStream.write(bytes, 0, read);              }              outpuStream.flush();              outpuStream.close();          } catch(IOException iox){              iox.printStackTrace();          } finally {              if(outpuStream != null){                  try{outpuStream.close();} catch(Exception ex){}              }          }          return "File Upload Successfully !!";      }  } | |

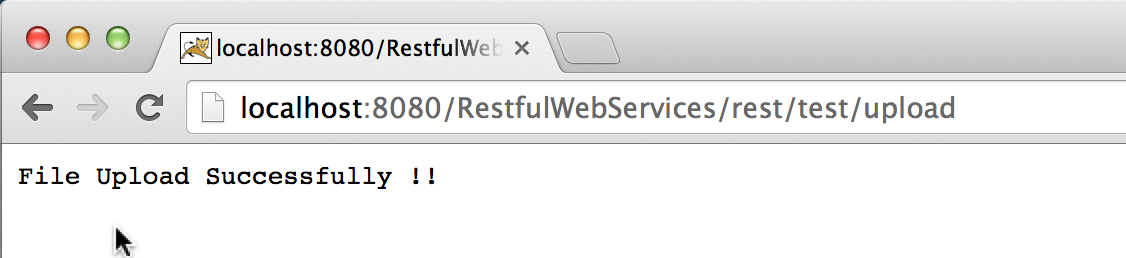
Try below URL to upload file:

**http://localhost:8080/RestfulWebServices/index.html**

**User Upload Form**



**Output**



# How to download file using java restful web services?

You need to do two stpes to download a file from java restful web services.

1) Annotate your service method with @Produces annotation. This annotation should have the file MIME type as a value. For example, if you are downloading pdf file then MIME type should be "application/pdf", incase if you are downloading png image file, then MIME type should be "image/png".  
2) In the Response header, set “Content-Disposition” details, which helps to prompt download box on browser.

In the previous examples we have given details of application setup, dependencies, web.xml file configurations: If you want to know about these configuration, please refer these:

|  |  |
| --- | --- |
| |  | | --- | | package com.java2novice.restful;    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("/download")  public class RestDownloadService {        @GET      @Path("/service-record")      @Produces("application/pdf")      public Response getFile() {            File file = new File("C:\java2novice\employee\_1415.pdf");            ResponseBuilder response = Response.ok((Object) file);          response.header("Content-Disposition",              "attachment; filename=\"employee\_1415.pdf\"");          return response.build();      }  } | |

Try below URL to download file:

**http://localhost:8080/RestfulWebServices/download/service-record**

# XML based Restful web service with RESTEasy and JAXB.

In this page you will see support for XML using JAXB and RESTEasy API. JAXB is used for mapping java classes to equivalent xml documents and vice versa. It is done using marshalling and and unmarshalling features of JAXB. In this example we will convert Order object to xml format.

We need resteasy-jaxb-provider.jar file to support xml file, here is the pom.xml file.

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/xml-resteasy-jaxb/)   |  |  | | --- | --- | | 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 | <?xml version="1.0" encoding="UTF-8"?>  <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>RestfulWebServices</groupId>      <artifactId>RestfulWebServices</artifactId>      <version>0.0.1-SNAPSHOT</version>      <packaging>war</packaging>      <repositories>          <repository>              <id>jboss</id>              <url>https://repository.jboss.org/nexus/content/groups/public-jboss/</url>          </repository>      </repositories>      <dependencies>          <dependency>              <groupId>org.jboss.resteasy</groupId>              <artifactId>resteasy-jaxb-provider</artifactId>              <version>2.3.7.Final</version>          </dependency>          <dependency>              <groupId>org.jboss.resteasy</groupId>              <artifactId>resteasy-jaxrs</artifactId>              <version>2.3.7.Final</version>          </dependency>      </dependencies>  </project> | |

Web.xml file for your reference:

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/xml-resteasy-jaxb/)   |  |  | | --- | --- | | 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 | <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">        <!-- Auto scan REST service -->      <context-param>          <param-name>resteasy.scan</param-name>          <param-value>true</param-value>      </context-param>        <listener>          <listener-class>              org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap          </listener-class>      </listener>        <servlet>          <servlet-name>resteasy-servlet</servlet-name>          <servlet-class>              org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher          </servlet-class>      </servlet>        <servlet-mapping>          <servlet-name>resteasy-servlet</servlet-name>          <url-pattern>/\*</url-pattern>      </servlet-mapping>  </web-app> | |

Our model class Order is annotated with required JAXB annoations to support xml transformation:

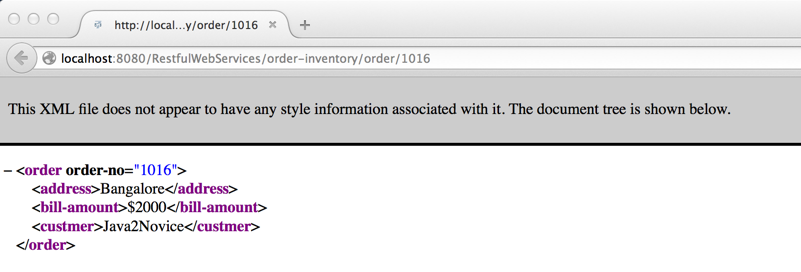
|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/xml-resteasy-jaxb/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | package com.javacoffee.model;    import javax.xml.bind.annotation.XmlAttribute;  import javax.xml.bind.annotation.XmlElement;  import javax.xml.bind.annotation.XmlRootElement;    @XmlRootElement(name = "order")  public class Order {          private int orderNo;      private String custmer;      private String address;      private String amount;        @XmlAttribute(name = "order-no")      public int getOrderNo() {          return orderNo;      }      public void setOrderNo(int orderNo) {          this.orderNo = orderNo;      }        @XmlElement      public String getCustmer() {          return custmer;      }      public void setCustmer(String custmer) {          this.custmer = custmer;      }        @XmlElement      public String getAddress() {          return address;      }      public void setAddress(String address) {          this.address = address;      }        @XmlElement(name = "bill-amount")      public String getAmount() {          return amount;      }      public void setAmount(String amount) {          this.amount = amount;      }    } | |

Note that our restful web service API returning xml, so annotate your service method with @Produces and specify MIME type as application/xml.

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/xml-resteasy-jaxb/)   |  |  | | --- | --- | | 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.javacoffee.restful;    import javax.ws.rs.GET;  import javax.ws.rs.Path;  import javax.ws.rs.PathParam;  import javax.ws.rs.Produces;  import com.java2novice.model.Order;    @Path("/order-inventory")  public class OrderInventoryService {        @GET      @Path("/order/{orderId}")      @Produces("application/xml")      public Order getUserById(@PathParam("orderId") Integer orderId){            Order ord = new Order();          ord.setOrderNo(orderId);          ord.setCustmer("Java2Novice");          ord.setAddress("Bangalore");          ord.setAmount("$2000");          return ord;      }  } | |

Try below URL to get xml output:

**http://localhost:8080/RestfulWebServices/order-inventory/order/1016**



# XML based Restful web service with Jersey and JAXB.

In this page you will see support for XML using JAXB and Jersey API. JAXB is used for mapping java classes to equivalent xml documents and vice versa. It is done using marshalling and and unmarshalling features of JAXB. In this example we will convert Order object to xml format.

Here is the pom.xml file.

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/xml-jersey-jaxb/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | <?xml version="1.0" encoding="UTF-8"?>  <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>RestfulWebServices</groupId>      <artifactId>RestfulWebServices</artifactId>      <version>0.0.1-SNAPSHOT</version>      <packaging>war</packaging>      <dependencies>          <dependency>              <groupId>com.sun.jersey</groupId>              <artifactId>jersey-server</artifactId>              <version>1.17</version>          </dependency>          <dependency>              <groupId>com.sun.jersey</groupId>              <artifactId>jersey-servlet</artifactId>              <version>1.17</version>          </dependency>      </dependencies>  </project> | |

Web.xml file for your reference:

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/xml-jersey-jaxb/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <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">      <servlet>          <servlet-name>jersey-serlvet</servlet-name>          <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>          <init-param>              <param-name>jersey.config.server.provider.packages</param-name>              <param-value>com.java2novice.restful</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> | |

Our model class Order is annotated with required JAXB annoations to support xml transformation:

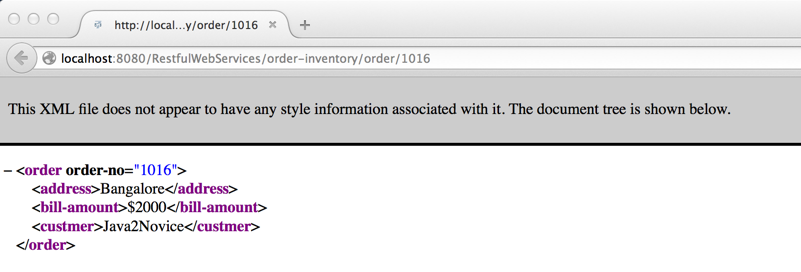
|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/xml-jersey-jaxb/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | package com.javacoffee.model;    import javax.xml.bind.annotation.XmlAttribute;  import javax.xml.bind.annotation.XmlElement;  import javax.xml.bind.annotation.XmlRootElement;    @XmlRootElement(name = "order")  public class Order {          private int orderNo;      private String custmer;      private String address;      private String amount;        @XmlAttribute(name = "order-no")      public int getOrderNo() {          return orderNo;      }      public void setOrderNo(int orderNo) {          this.orderNo = orderNo;      }        @XmlElement      public String getCustmer() {          return custmer;      }      public void setCustmer(String custmer) {          this.custmer = custmer;      }        @XmlElement      public String getAddress() {          return address;      }      public void setAddress(String address) {          this.address = address;      }        @XmlElement(name = "bill-amount")      public String getAmount() {          return amount;      }      public void setAmount(String amount) {          this.amount = amount;      }    } | |

Note that our restful web service API returning xml, so annotate your service method with @Produces and specify MIME type as application/xml.

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/xml-jersey-jaxb/)   |  |  | | --- | --- | | 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 | package com.javacoffee.restful;    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 com.java2novice.model.Order;    @Path("/order-inventory")  public class OrderInventoryService {        @GET      @Path("/order/{orderId}")      @Produces(MediaType.APPLICATION\_XML)      public Order getUserById(@PathParam("orderId") Integer orderId){            Order ord = new Order();          ord.setOrderNo(orderId);          ord.setCustmer("Java2Novice");          ord.setAddress("Bangalore");          ord.setAmount("$2000");          return ord;      }  } | |

Try below URL to get xml output:

**http://localhost:8080/RestfulWebServices/order-inventory/order/1016**



# Json based Restful web service with RESTEasy and Jackson.

In this page you will see support for Json using RESTEasy and Jackson APIs. Jackson is is a multi-purpose Java library for processing JSON data format. Jackson aims to be the best possible combination of fast, correct, lightweight, and ergonomic for developers In this example we will convert Order object to json format.

Here is the pom.xml file.

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/json-resteasy-jackson/)   |  |  | | --- | --- | | 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 | <?xml version="1.0" encoding="UTF-8"?>  <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>RestfulWebServices</groupId>      <artifactId>RestfulWebServices</artifactId>      <version>0.0.1-SNAPSHOT</version>      <packaging>war</packaging>      <repositories>          <repository>              <id>jboss</id>              <url>https://repository.jboss.org/nexus/content/groups/public-jboss/</url>          </repository>      </repositories>      <dependencies>          <dependency>              <groupId>org.jboss.resteasy</groupId>              <artifactId>resteasy-jackson-provider</artifactId>              <version>2.3.7.Final</version>          </dependency>          <dependency>              <groupId>org.jboss.resteasy</groupId>              <artifactId>resteasy-jaxrs</artifactId>              <version>2.3.7.Final</version>          </dependency>      </dependencies>  </project> | |

Web.xml file for your reference:

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/json-resteasy-jackson/)   |  |  | | --- | --- | | 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 | <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">       <!-- Auto scan REST service -->      <context-param>          <param-name>resteasy.scan</param-name>          <param-value>true</param-value>      </context-param>      <listener>          <listener-class>              org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap          </listener-class>      </listener>      <servlet>          <servlet-name>resteasy-servlet</servlet-name>          <servlet-class>              org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher          </servlet-class>      </servlet>      <servlet-mapping>          <servlet-name>resteasy-servlet</servlet-name>          <url-pattern>/\*</url-pattern>      </servlet-mapping>  </web-app> | |

Our model class Order is annotated with required jackson annoations to support json transformation:

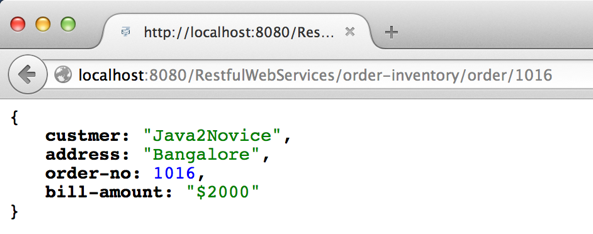
|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/json-resteasy-jackson/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | package com.javacoffee.model;    import org.codehaus.jackson.annotate.JsonProperty;    public class Order {        @JsonProperty("order-no")      private int orderNo;        @JsonProperty      private String custmer;        private String address;        @JsonProperty("bill-amount")      private String amount;          public int getOrderNo() {          return orderNo;      }      public void setOrderNo(int orderNo) {          this.orderNo = orderNo;      }        public String getCustmer() {          return custmer;      }      public void setCustmer(String custmer) {          this.custmer = custmer;      }        public String getAddress() {          return address;      }      public void setAddress(String address) {          this.address = address;      }        public String getAmount() {          return amount;      }      public void setAmount(String amount) {          this.amount = amount;      }    } | |

Remember that our restful web service API returning json, so annotate your service method with @Produces and specify MIME type as application/jon.

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/json-resteasy-jackson/)   |  |  | | --- | --- | | 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 | package com.javacoffee.restful;    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 com.java2novice.model.Order;    @Path("/order-inventory")  public class OrderInventoryService {        @GET      @Path("/order/{orderId}")      @Produces(MediaType.APPLICATION\_JSON)      public Order getUserById(@PathParam("orderId") Integer orderId){            Order ord = new Order();          ord.setOrderNo(orderId);          ord.setCustmer("Java2Novice");          ord.setAddress("Bangalore");          ord.setAmount("$2000");          return ord;      }  } | |

Try below URL to get xml output:

**http://localhost:8080/RestfulWebServices/order-inventory/order/1016**



# son based Restful web service with Jersey and Jackson.

In this page you will see support for Json using Jersey and Jackson APIs. Jackson is is a multi-purpose Java library for processing JSON data format. Jackson aims to be the best possible combination of fast, correct, lightweight, and ergonomic for developers In this example we will convert Order object to json format.

Here is the pom.xml file. You need jersey-json jar file.

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/json-jersey-jackson/)   |  |  | | --- | --- | | 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 | <?xml version="1.0" encoding="UTF-8"?>  <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>RestfulWebServices</groupId>      <artifactId>RestfulWebServices</artifactId>      <version>0.0.1-SNAPSHOT</version>      <packaging>war</packaging>      <dependencies>          <dependency>              <groupId>com.sun.jersey</groupId>              <artifactId>jersey-server</artifactId>              <version>1.17</version>          </dependency>          <dependency>              <groupId>com.sun.jersey</groupId>              <artifactId>jersey-servlet</artifactId>              <version>1.17</version>          </dependency>          <dependency>              <groupId>com.sun.jersey</groupId>              <artifactId>jersey-json</artifactId>              <version>1.17</version>          </dependency>      </dependencies>  </project> | |

Web.xml file for your reference. In web.xml add “com.sun.jersey.api.json.POJOMappingFeature” as “init-param” which supports Json object mapping.

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/json-jersey-jackson/)   |  |  | | --- | --- | | 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 | <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">       <servlet>          <servlet-name>jersey-serlvet</servlet-name>          <servlet-class>             com.sun.jersey.spi.container.servlet.ServletContainer          </servlet-class>          <init-param>              <param-name>jersey.config.server.provider.packages</param-name>              <param-value>com.java2novice.restful</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>/\*</url-pattern>      </servlet-mapping>  </web-app> | |

Our model class Order is annotated with required jackson annoations to support json transformation:

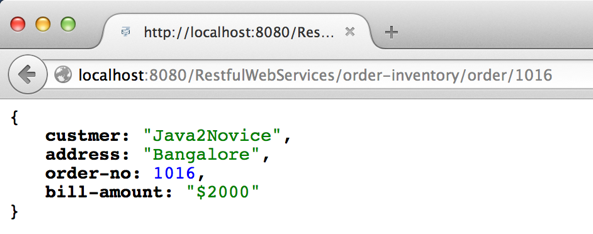
|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/json-jersey-jackson/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | package com.javacoffee.model;    import org.codehaus.jackson.annotate.JsonProperty;    public class Order {        @JsonProperty("order-no")      private int orderNo;        @JsonProperty      private String custmer;        private String address;        @JsonProperty("bill-amount")      private String amount;          public int getOrderNo() {          return orderNo;      }      public void setOrderNo(int orderNo) {          this.orderNo = orderNo;      }        public String getCustmer() {          return custmer;      }      public void setCustmer(String custmer) {          this.custmer = custmer;      }        public String getAddress() {          return address;      }      public void setAddress(String address) {          this.address = address;      }        public String getAmount() {          return amount;      }      public void setAmount(String amount) {          this.amount = amount;      }    } | |

Remember that our restful web service API returning json, so annotate your service method with @Produces and specify MIME type as application/jon.

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/json-jersey-jackson/)   |  |  | | --- | --- | | 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 | package com.javacoffee.restful;    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 com.java2novice.model.Order;    @Path("/order-inventory")  public class OrderInventoryService {        @GET      @Path("/order/{orderId}")      @Produces(MediaType.APPLICATION\_JSON)      public Order getUserById(@PathParam("orderId") Integer orderId){            Order ord = new Order();          ord.setOrderNo(orderId);          ord.setCustmer("Java2Novice");          ord.setAddress("Bangalore");          ord.setAmount("$2000");          return ord;      }  } | |

Try below URL to get xml output:

**http://localhost:8080/RestfulWebServices/order-inventory/order/1016**



|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| How to input json request with Jersey and Jackson? In this page you will see support for Json using Jersey and Jackson APIs. Jackson is is a multi-purpose Java library for processing JSON data format. Jackson aims to be the best possible combination of fast, correct, lightweight, and ergonomic for developers In this example we will send json as a input, and the json request will be mapped to Order object.  Here is the pom.xml file. You need jersey-json jar file.   |  |  |  | | --- | --- | --- | | [?](http://www.java2novice.com/restful-web-services/json-input-jersey-jackson/)   |  |  | | --- | --- | | 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 | <?xml version="1.0" encoding="UTF-8"?>  <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>RestfulWebServices</groupId>      <artifactId>RestfulWebServices</artifactId>      <version>0.0.1-SNAPSHOT</version>      <packaging>war</packaging>      <dependencies>          <dependency>              <groupId>com.sun.jersey</groupId>              <artifactId>jersey-server</artifactId>              <version>1.17</version>          </dependency>          <dependency>              <groupId>com.sun.jersey</groupId>              <artifactId>jersey-servlet</artifactId>              <version>1.17</version>          </dependency>          <dependency>              <groupId>com.sun.jersey</groupId>              <artifactId>jersey-json</artifactId>              <version>1.17</version>          </dependency>      </dependencies>  </project> | |   Web.xml file for your reference. In web.xml add “com.sun.jersey.api.json.POJOMappingFeature” as “init-param” which supports Json object mapping.   |  |  |  | | --- | --- | --- | | [?](http://www.java2novice.com/restful-web-services/json-input-jersey-jackson/)   |  |  | | --- | --- | | 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 | <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">       <servlet>          <servlet-name>jersey-serlvet</servlet-name>          <servlet-class>             com.sun.jersey.spi.container.servlet.ServletContainer          </servlet-class>          <init-param>              <param-name>jersey.config.server.provider.packages</param-name>              <param-value>com.java2novice.restful</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>/\*</url-pattern>      </servlet-mapping>  </web-app> | |   Our model class Order is annotated with required jackson annoations to support json transformation:   |  |  |  | | --- | --- | --- | | [?](http://www.java2novice.com/restful-web-services/json-input-jersey-jackson/)   |  |  | | --- | --- | | 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 | package com.javacoffee.model;    import org.codehaus.jackson.annotate.JsonProperty;    public class Order {        @JsonProperty      private String custmer;        private String address;        @JsonProperty("bill-amount")      private String amount;        public String getCustmer() {          return custmer;      }      public void setCustmer(String custmer) {          this.custmer = custmer;      }        public String getAddress() {          return address;      }      public void setAddress(String address) {          this.address = address;      }        public String getAmount() {          return amount;      }      public void setAmount(String amount) {          this.amount = amount;      }    } | |   Remember that our restful web service API accepting json as an input, we should annotate our service method with @Consumes and specify MIME type as application/jon. Closely watch our service method input parameter, it is of type Order, before calling our service method, the json is mapped to Order object.   |  |  |  | | --- | --- | --- | | [?](http://www.java2novice.com/restful-web-services/json-input-jersey-jackson/)   |  |  | | --- | --- | | 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.javacoffee.restful;    import javax.ws.rs.Consumes;  import javax.ws.rs.POST;  import javax.ws.rs.Path;  import javax.ws.rs.Produces;  import javax.ws.rs.core.MediaType;  import javax.ws.rs.core.Response;    import com.javacoffee.model.Order;        @POST      @Path("/order")      @Consumes(MediaType.APPLICATION\_JSON)      public Response getUserById(Order inputOrder){            System.out.println("Received order from :"+inputOrder.getCustmer());          System.out.println("Order worth: "+inputOrder.getAmount());          System.out.println("Customer address: "+inputOrder.getAddress());            return Response.status(200).entity("Your order is in-progress").build();      }  } | |   Here is the json requst and response:  http://www.java2novice.com/images/json_input.png  The console output: |
| |  | | --- | | Output: | | [stdout] (http--127.0.0.1-8080-1) Received order from :Java2Novice  [stdout] (http--127.0.0.1-8080-1) Order worth: $2000  [stdout] (http--127.0.0.1-8080-1) Customer address: Bangalore | |

# Java client for restful web service using java.net package

In this page you will come to know how to create java client for restful web services using java.net package. We will have two sections here, the first section talks about how to connect to "GET" request, and the second section shows how to connect to "POST" type of requests.

## Java Client for GET Request

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/java-client-net-url/)   |  |  | | --- | --- | | 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 | package com.javacoffee.rest.client;    import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStream;  import java.io.InputStreamReader;  import java.net.HttpURLConnection;  import java.net.MalformedURLException;  import java.net.URL;    public class RestJavaNetClient {        public static void main(String a[]){            String url = "<http://localhost:8080/RestfulWebServices/order-inventory/order/1016>";          HttpURLConnection urlConn = null;          BufferedReader reader = null;          try {              URL urlObj = new URL(url);              urlConn = (HttpURLConnection) urlObj.openConnection();              urlConn.setRequestMethod("GET");              urlConn.setConnectTimeout(5000);              urlConn.setReadTimeout(5000);              urlConn.setRequestProperty("Accept", "application/json");              if (urlConn.getResponseCode() != HttpURLConnection.HTTP\_OK) {                  System.err.println("Unable to connect to the URL...");                  return;              }              System.out.println("Connected to the server...");              InputStream is = urlConn.getInputStream();              reader = new BufferedReader(new InputStreamReader((is)));              System.out.println("Reading data from server...");              String tmpStr = null;              while((tmpStr = reader.readLine()) != null){                  System.out.println(tmpStr);              }          } catch (MalformedURLException e) {              // TODO Auto-generated catch block              e.printStackTrace();          } catch (IOException e) {              // TODO Auto-generated catch block              e.printStackTrace();          } finally {              try {                  if(reader != null) reader.close();                  if(urlConn != null) urlConn.disconnect();              } catch(Exception ex){                }          }      }  } | |

The output for above code is:

|  |
| --- |
| Output: |
| Connected to the server...  Reading data from server...  {"custmer":"Java2Novice","address":"Bangalore","bill-amount":"$2000"} |

## Java Client for POST Request

In this example you will see how to send json input through java client using POST method:

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/java-client-net-url/)   |  |  | | --- | --- | | 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  56  57  58  59  60  61 | package com.javacoffee.rest.client;    import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStream;  import java.io.InputStreamReader;  import java.io.OutputStream;  import java.net.HttpURLConnection;  import java.net.MalformedURLException;  import java.net.URL;    public class PostJavaNetClient {        public static void main(String a[]){            String url = "<http://localhost:8080/RestfulWebServices/order-inventory/order>";          HttpURLConnection urlConn = null;          BufferedReader reader = null;          OutputStream ouputStream = null;          String jsonInput = "{\"custmer\":\"Java2novice\",\"address\":\"Bangalore\","+                              "\"bill-amount\":\"$2000\"}";          try {              URL urlObj = new URL(url);              urlConn = (HttpURLConnection) urlObj.openConnection();              urlConn.setDoOutput(true);              urlConn.setRequestMethod("POST");              urlConn.setRequestProperty("Content-Type", "application/json");              urlConn.setConnectTimeout(5000);              urlConn.setReadTimeout(5000);              urlConn.setRequestProperty("Accept", "application/json");              // send json input request              ouputStream = urlConn.getOutputStream();              ouputStream.write(jsonInput.getBytes());              ouputStream.flush();              if (urlConn.getResponseCode() != HttpURLConnection.HTTP\_OK) {                  System.err.println("Unable to connect to the URL...");                  return;              }              System.out.println("Connected to the server...");              InputStream is = urlConn.getInputStream();              reader = new BufferedReader(new InputStreamReader((is)));              String tmpStr = null;              while((tmpStr = reader.readLine()) != null){                  System.out.println(tmpStr);              }          } catch (MalformedURLException e) {              // TODO Auto-generated catch block              e.printStackTrace();          } catch (IOException e) {              // TODO Auto-generated catch block              e.printStackTrace();          } finally {              try {                  if(reader != null) reader.close();                  if(urlConn != null) urlConn.disconnect();              } catch(Exception ex){                }          }      }  } | |

The output for above code is:

|  |
| --- |
| Output: |
| Connected to the server...  Your order is in-progress |

# Java client for restful web service using Jersey API

In this page you will come to know how to create java client for restful web services using Jersey API. You will see two sections here, the first section talks about how to connect to "GET" request, and the second section shows how to connect to "POST" type of requests.

## Java Client for GET Request using Jersey API

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/java-client-jersey-api/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package com.javacoffee.rest.client;    import com.sun.jersey.api.client.Client;  import com.sun.jersey.api.client.ClientResponse;  import com.sun.jersey.api.client.WebResource;    public class JersyGetClient {        public static void main(String a[]){            String url = "<http://localhost:8080/RestfulWebServices/order-inventory/order/1016>";          Client restClient = Client.create();          WebResource webResource = restClient.resource(url);          ClientResponse resp = webResource.accept("application/json")                                                      .get(ClientResponse.class);          if(resp.getStatus() != 200){              System.err.println("Unable to connect to the server");          }          String output = resp.getEntity(String.class);          System.out.println("response: "+output);      }  } | |

The output for above code is:

|  |
| --- |
| Output: |
| response: {"custmer":"Java2Novice","address":"Bangalore","bill-amount":"$2000"} |

## Java Client for POST Request using Jersey API

In this example you will see how to send json input through java client using POST method:

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/restful-web-services/java-client-jersey-api/)   |  |  | | --- | --- | | 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.javacoffee.rest.client;    import com.sun.jersey.api.client.Client;  import com.sun.jersey.api.client.ClientResponse;  import com.sun.jersey.api.client.WebResource;    public class JerseyPostClient {        public static void main(String a[]){            String url = "<http://localhost:8080/RestfulWebServices/order-inventory/order>";          String jsonInput = "{\"custmer\":\"Java2novice\",\"address\":\"Bangalore\","+                  "\"bill-amount\":\"$2000\"}";          Client restClient = Client.create();          WebResource webResource = restClient.resource(url);          ClientResponse resp = webResource.type("application/json")                                      .post(ClientResponse.class, jsonInput);          if(resp.getStatus() != 200){              System.err.println("Unable to connect to the server");          }          String output = resp.getEntity(String.class);          System.out.println("response: "+output);      }  } | |

The output for above code is:

|  |
| --- |
| Output: |
| response: Your order is in-progress |
| Java restful webservices with HTTP basic authentication.  In the context of a HTTP transaction, basic access authentication is a method for an HTTP user agent to provide a user name and password when making a request.  HTTP Basic authentication implementation is the simplest technique for enforcing access controls to web resources because it doesn't require cookies, session identifier and login pages. Rather, HTTP Basic authentication uses static, standard HTTP headers which means that no handshakes have to be done in anticipation.  When the user agent wants to send the server authentication credentials it may use the Authorization header. The Authorization header is constructed as follows:  1) Username and password are combined into a string "username:password" 2) The resulting string is then encoded using Base64 encoding 3) The authorization method and a space i.e. "Basic " is then put before the encoded string.  For example, if the user agent uses 'Aladdin' as the username and 'open sesame' as the password then the header is formed as follows:   |  | | --- | | Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ== |   Java Rest Service method with GET Request which supports HTTP basic authentication   |  |  |  | | --- | --- | --- | | [?](http://www.java2novice.com/restful-web-services/http-basic-authentication/)   |  |  | | --- | --- | | 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  56  57  58  59  60  61 | package com.javacoffee.restful;    import java.io.IOException;    import javax.ws.rs.GET;  import javax.ws.rs.HeaderParam;  import javax.ws.rs.Path;  import javax.ws.rs.PathParam;  import javax.ws.rs.Produces;  import javax.ws.rs.core.MediaType;  import sun.misc.BASE64Decoder;    import com.java2novice.model.Order;    @Path("/order-inventory")  public class OrderInventoryService {        @GET      @Path("/order/{orderId}")      @Produces(MediaType.APPLICATION\_JSON)      public Object getUserById(@PathParam("orderId") Integer orderId,                              @HeaderParam("authorization") String authString){            if(!isUserAuthenticated(authString)){              return "{\"error\":\"User not authenticated\"}";          }          Order ord = new Order();          ord.setCustmer("Java2Novice");          ord.setAddress("Bangalore");          ord.setAmount("$2000");          return ord;      }        private boolean isUserAuthenticated(String authString){            String decodedAuth = "";          // Header is in the format "Basic 5tyc0uiDat4"          // We need to extract data before decoding it back to original string          String[] authParts = authString.split("\\s+");          String authInfo = authParts[1];          // Decode the data back to original string          byte[] bytes = null;          try {              bytes = new BASE64Decoder().decodeBuffer(authInfo);          } catch (IOException e) {              // TODO Auto-generated catch block              e.printStackTrace();          }          decodedAuth = new String(bytes);          System.out.println(decodedAuth);            /\*\*           \* here you include your logic to validate user authentication.           \* it can be using ldap, or token exchange mechanism or your           \* custom authentication mechanism.           \*/          // your validation code goes here....            return true;      }  } | |   Java Client for GET Request using Jersey API with HTTP basic authentication   |  |  |  | | --- | --- | --- | | [?](http://www.java2novice.com/restful-web-services/http-basic-authentication/)   |  |  | | --- | --- | | 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 | package com.javacoffee.rest.client;    import sun.misc.BASE64Encoder;    import com.sun.jersey.api.client.Client;  import com.sun.jersey.api.client.ClientResponse;  import com.sun.jersey.api.client.WebResource;    public class JersyGetClient {        public static void main(String a[]){            String url = "<http://localhost:8080/RestfulWebServices/order-inventory/order/1016>";          String name = "java2novice";          String password = "Simple4u!";          String authString = name + ":" + password;          String authStringEnc = new BASE64Encoder().encode(authString.getBytes());          System.out.println("Base64 encoded auth string: " + authStringEnc);          Client restClient = Client.create();          WebResource webResource = restClient.resource(url);          ClientResponse resp = webResource.accept("application/json")                                           .header("Authorization", "Basic " + authStringEnc)                                           .get(ClientResponse.class);          if(resp.getStatus() != 200){              System.err.println("Unable to connect to the server");          }          String output = resp.getEntity(String.class);          System.out.println("response: "+output);      }  } | | |
| |  | | --- | |  | |