Developing Client for RESTful Web Service

{scrollbar}

This tutorial will take you through the steps required in developing, deploying and testing a RESTful Web Service Client in Apache Geronimo for a web services which are already deployed on the server

We will be creating a Web based Client which can access the RESTful web service via GET and POST methods. You can easily create this client without any external tools except that of a server environment.

To run this tutorial, as a minimum you will be required to have installed the following prerequisite software.

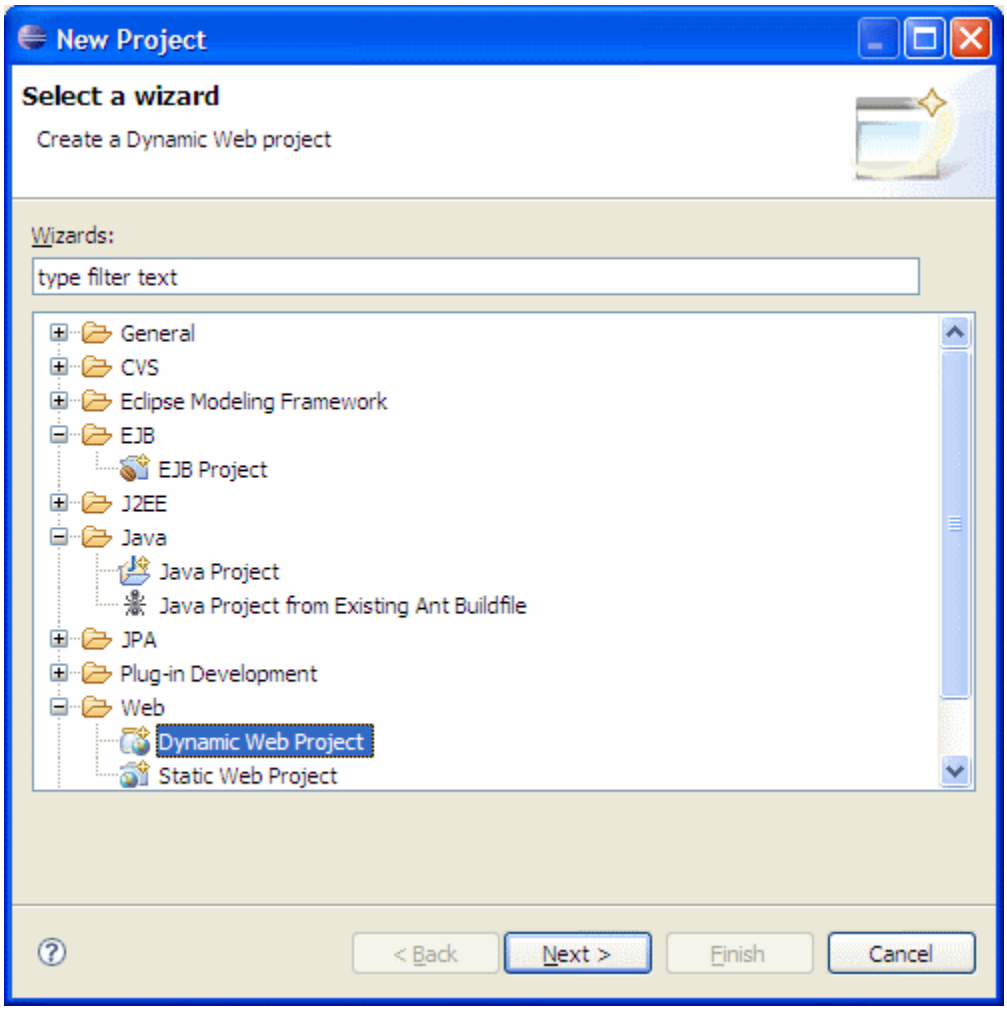
- Sun JDK 5.0+ (J2SE 1.5)
- Apache Geronimo 2.x
- Eclipse IDE for Java EE Developers - Europa release
- Geronimo Eclipse Plug-in 2.x

Details on installing Eclipse are provided in the Development environment section.
Deployed Web Service
This tutorial assumes that you have completed the Developing a simple RESTful Service tutorial. Here we will try to develop the client for the web service deployed in the above mentioned tutorial.

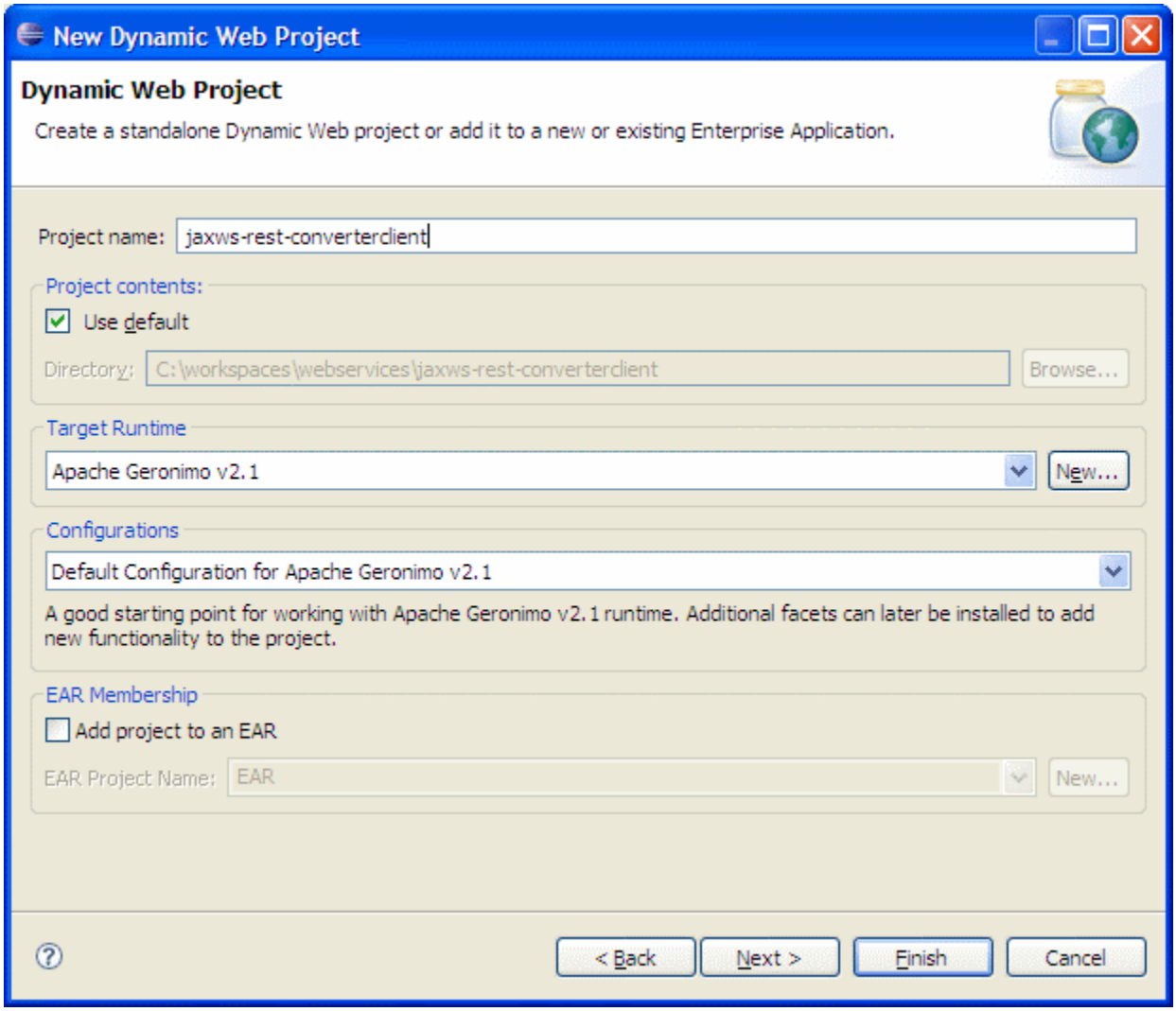
This tutorial will take you through the following steps:
2listpipe

Create a Dynamic Web Project to consume the Web Service

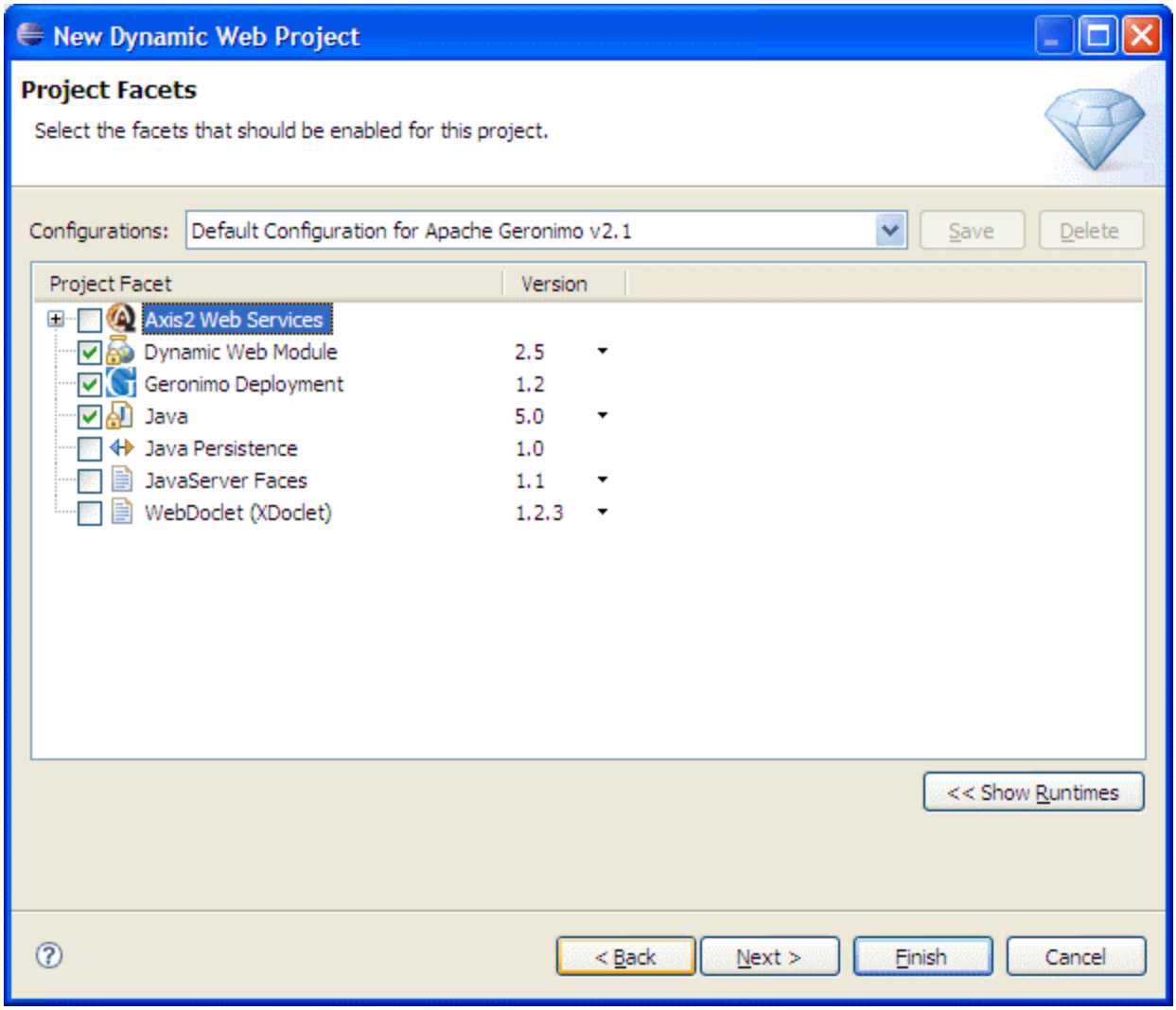
- From Eclipse main menu, select **File->New->Other**
- In the **New** dialog, select **Web->Dynamic Web Project** and click **Next**



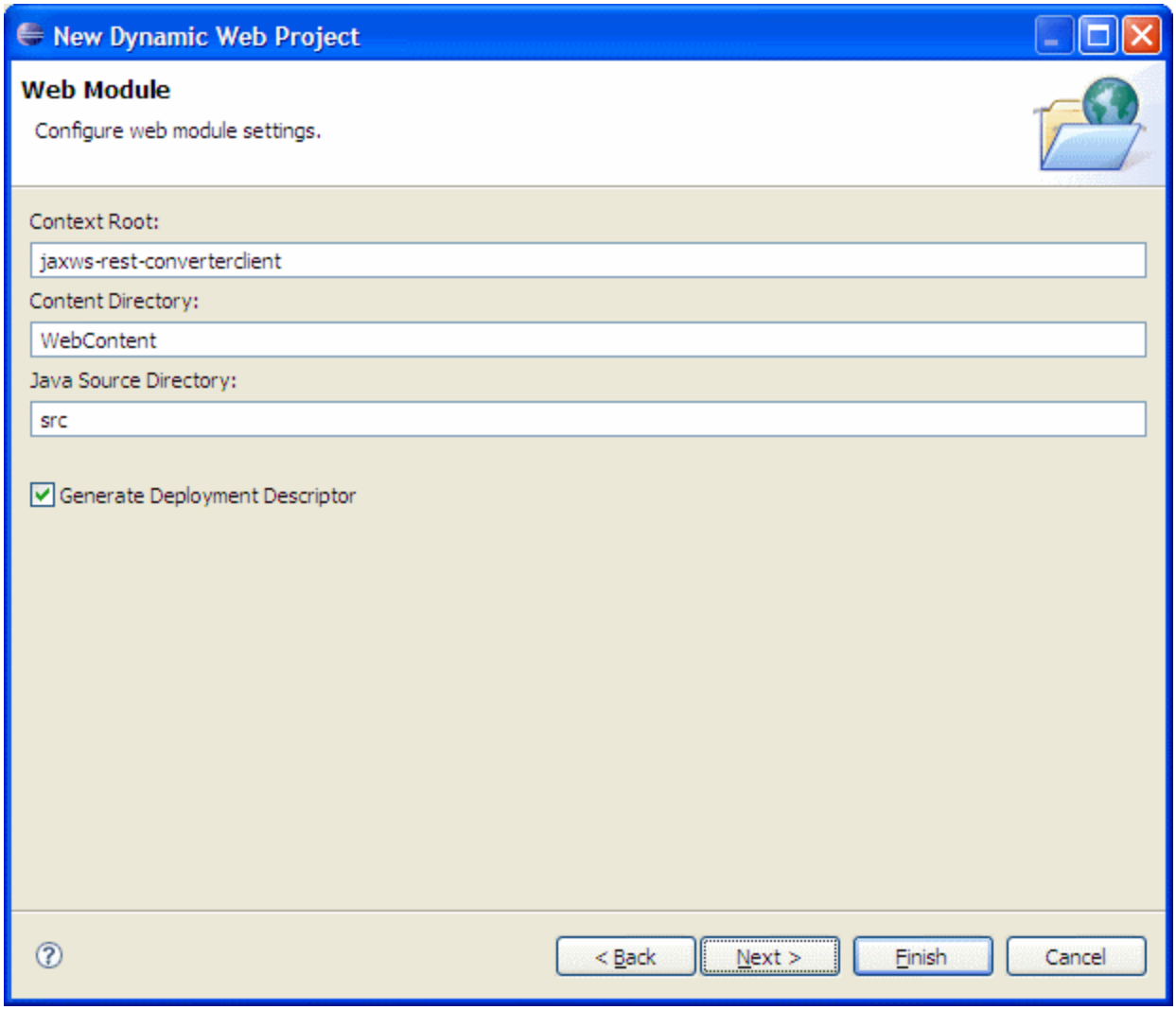
- Type **jaxws-rest-converterclient** as the **Project Name** and click **Next**



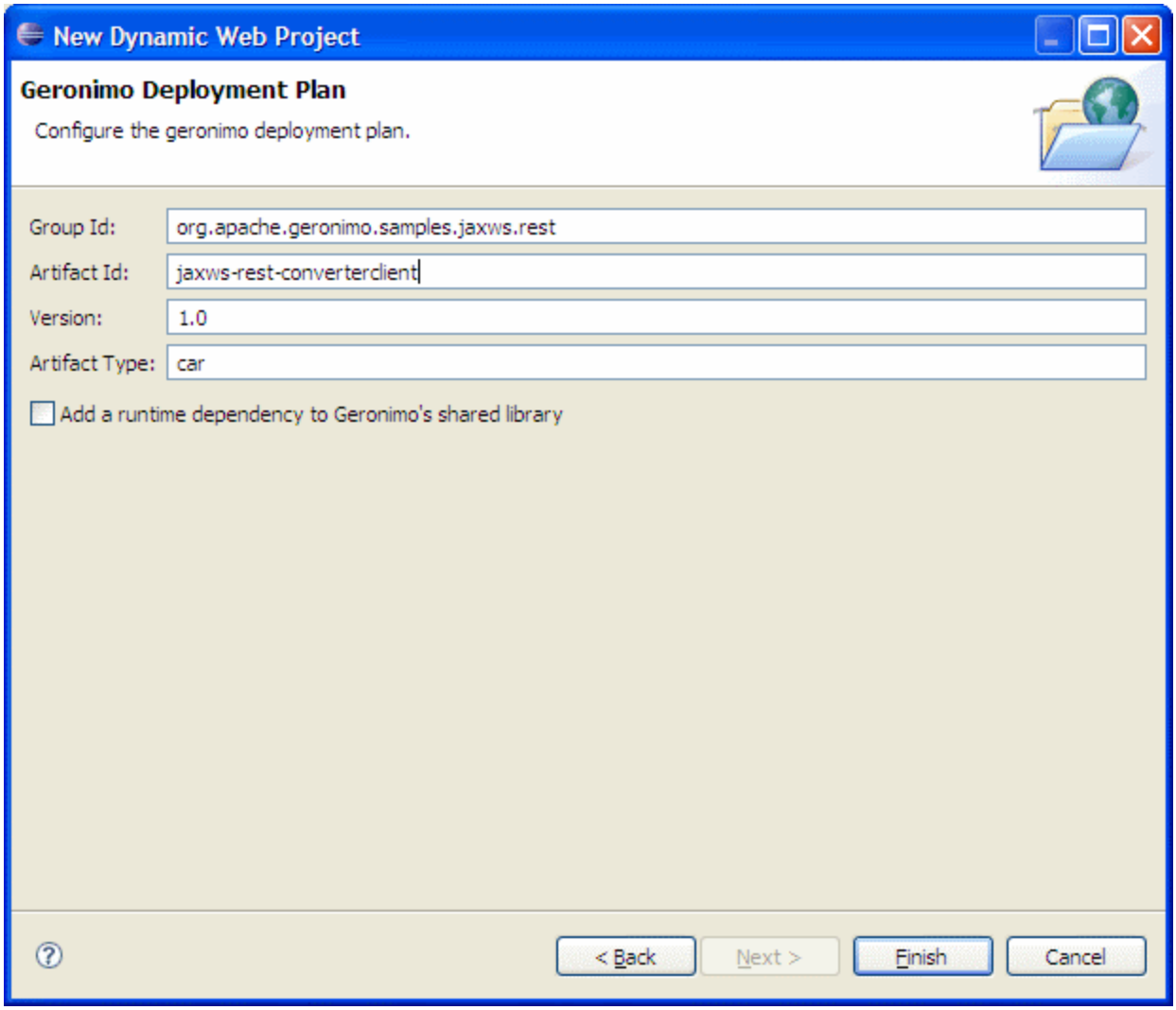
- On the **Project Facets** page, the default selections are enough.



- Make sure that the check box **Generate Deployment Descriptor** is selected and click **Next**



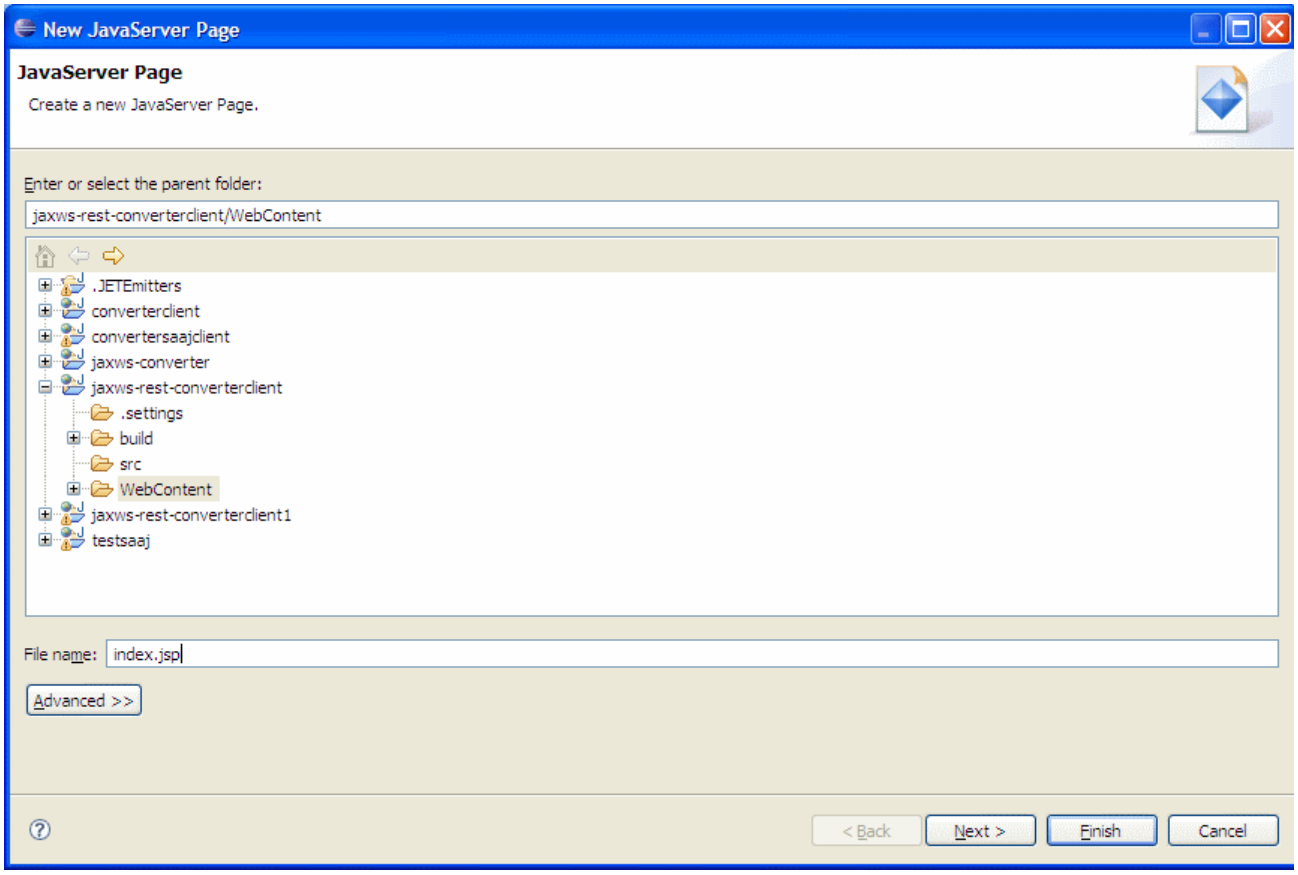
- On the **Geronimo Deployment Page** modify the **Group Id** to **org.apache.geronimo.samples.jaxws.rest** and the **Artifact Id** to **jaxws-rest-converterclient**.



- Click **Finish**

Developing the Web based Client

- Right Click the **jaxws-converterclient**, and Select **New->JSP**
- Name the jsp as **index.jsp** and click **Finish**

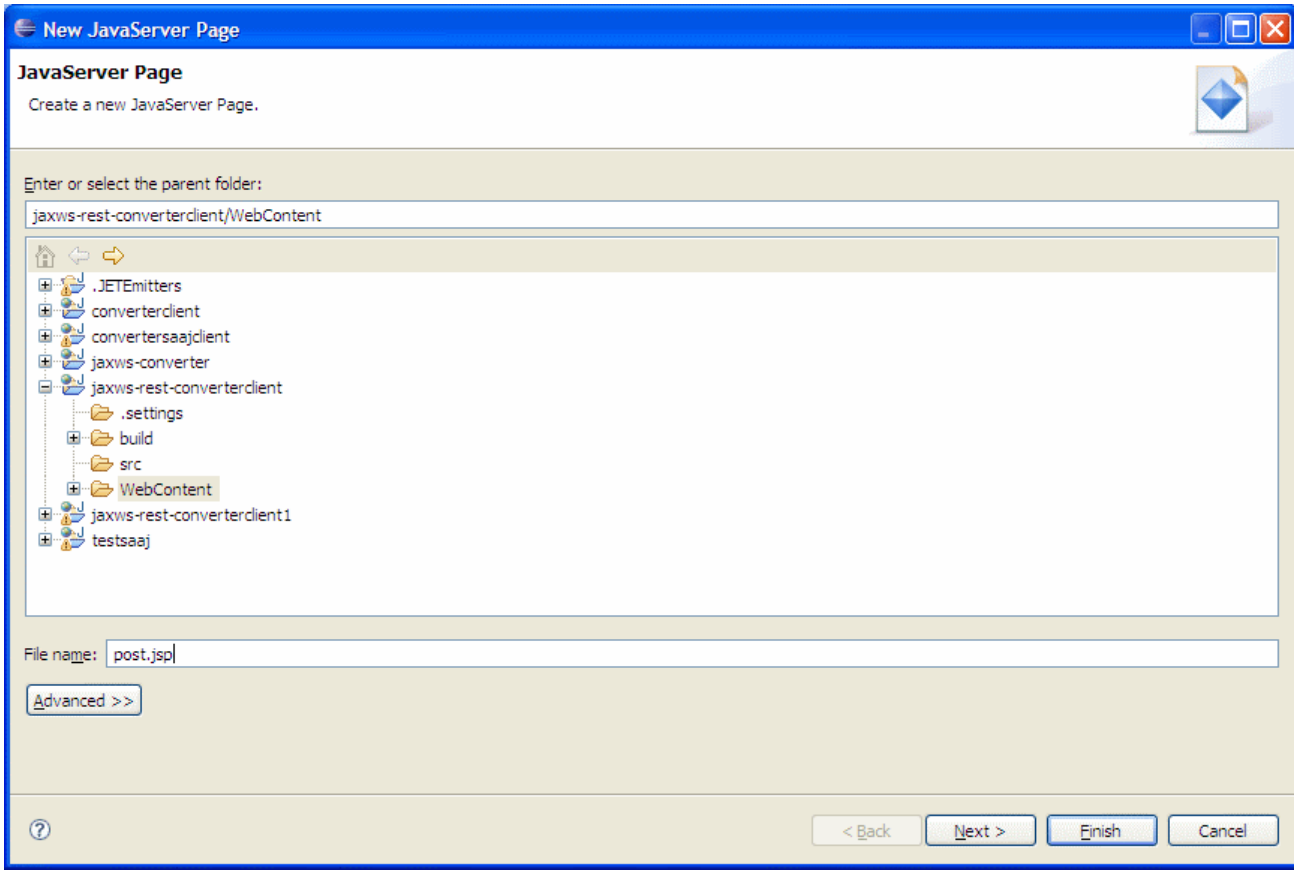


- Add the following code to the **index.jsp**

index.jspsolid <%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%> <!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=ISO-8859-1"> <title>RESTful Client</title> </head> <body> <h1>RESTful Client</h1>

 Test via Post <table bgcolor="#CCC"> <tr><td> <form action="index.jsp" >Dollars: <input type="text" name="query"> <input type="submit" value="Submit"></form> </td></tr> </table> <jsp:include page="/ConverterHandler" /> </body> </html>

- Right click again and add another jsp named **post.jsp**

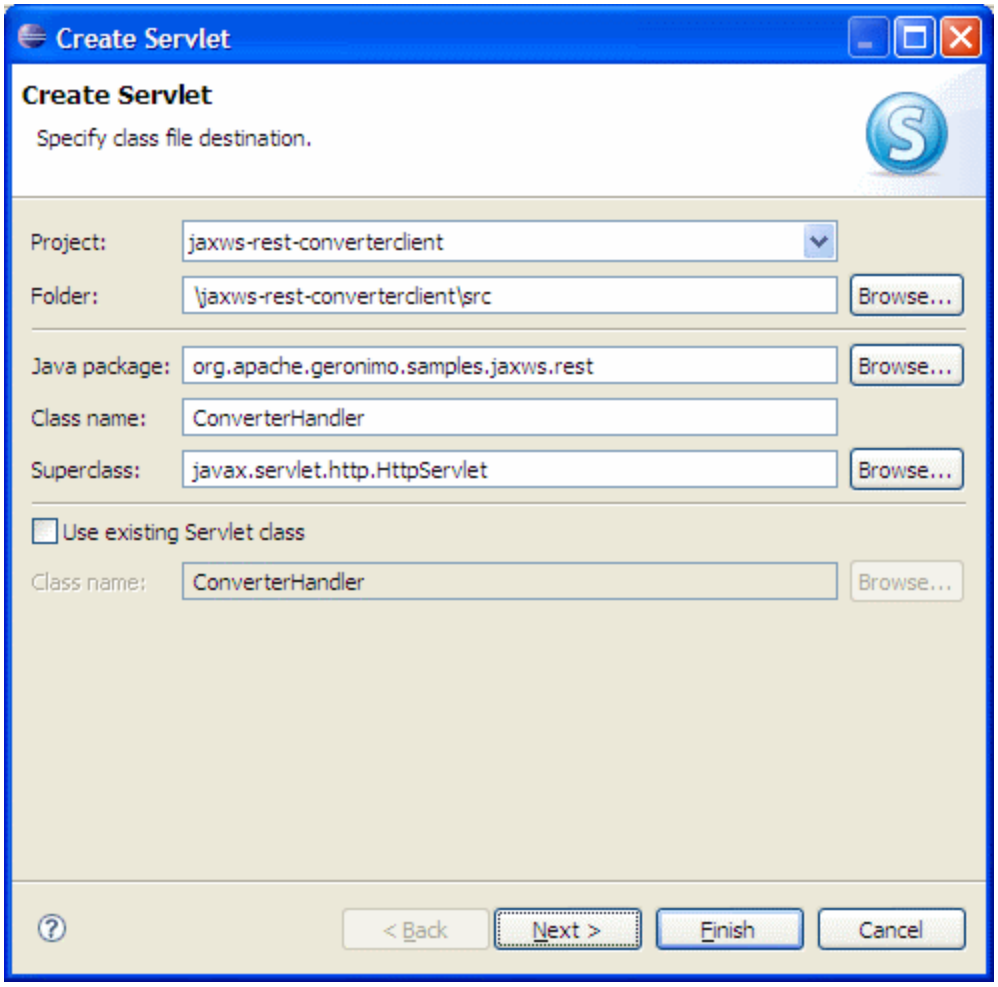


- Add the following code to **post.jsp**

post.jspsolid <%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%> <!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=ISO-8859-1"> <title>RESTful Client</title> </head> <body> <h1>RESTful Client</h1> Test via Get

 <table bgcolor="#CCC"> <tr><td> <form method="post" enctype=multipart/form-data action="post.jsp" >Enter full file Name: <input type="file" name="query"> <input type="submit" value="Submit"></form> </td></tr> </table> <jsp:include page="/ConverterHandler" /> </body> </html>

- Right click again and add a **Servlet** named **ConverterHandler**



- Add the following code to **ConverterHandler.java**

ConverterHandler.javasolid package org.apache.geronimo.samples.jaxws.rest; import java.io.*; import javax.servlet.ServletException; import javax.servlet.http.*; import javax.xml.parsers.*; import javax.xml.xpath.*; import org.apache.commons.httpclient.*; import org.apache.commons.httpclient.methods.*; import org.w3c.dom.*; import org.xml.sax.SAXException; public class ConverterHandler extends javax.servlet.http.HttpServlet implements javax.servlet.Servlet { static final long serialVersionUID = 1L; public ConverterHandler() { super(); } protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { PrintWriter out = response.getWriter(); String query = request.getParameter("query"); if (query != null && query.length() > 0) { out.println(" "); out.println("

"); String requestquery = "http://localhost:8080/jaxws-rest-converter/converter?amount=" + query; HttpClient client = new HttpClient(); GetMethod method = new GetMethod(requestquery); int statusCode = client.executeMethod(method); if (statusCode != HttpStatus.SC_OK) { System.err.println("Method failed: " + method.getStatusLine()); } InputStream rstream = null; rstream = method.getResponseBodyAsStream(); Document queryresponse = null; try { queryresponse = DocumentBuilderFactory.newInstance() .newDocumentBuilder().parse(rstream); } catch (SAXException e) { e.printStackTrace(); } catch (ParserConfigurationException e) { e.printStackTrace(); } XPathFactory factory = XPathFactory.newInstance(); XPath xPath = factory.newXPath(); NodeList nodes = null; try { nodes = (NodeList) xPath.evaluate("/return", queryresponse, XPathConstants.NODESET); } catch (XPathExpressionException e) { e.printStackTrace(); } int nodeCount = nodes.getLength(); for (int i = 0; i < nodeCount; i++) { // Get each xpath expression as a string String rupees = null; String euros = null; try { rupees = (String) xPath.evaluate("dollarToRupeesResponse", nodes.item(i), XPathConstants.STRING); euros = (String) xPath.evaluate("rupeesToEurosResponse", nodes.item(i), XPathConstants.STRING); } catch (XPathExpressionException e) { e.printStackTrace(); } out.println(query + " Dollars equals to " + rupees + " Rupees"); out.println("
"); out.println(rupees + " Rupees equals to " + euros + " Euros"); } out.println(" "); } } protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { // Has to be Completed } } Errors in Servlet
Here the servlet might notify that some imports are not resolved. We need to add two external jar files to resolve these errors.
The jar files that needed to be added in the build path are:
commons-codec-1.3.jar - <GERONIMO_INSTALL_DIR>\repository\commons-codec\commons-codec\1.3\commons-codec-1.3.jar
commons-httpclient-3.0.1.jar - <GERONIMO_INSTALL_DIR>\repository\commons-httpclient\commons-httpclient\3.0.1\commons-httpclient-3.0.1.jar

- Let us walkthrough the code of **ConverterHandler** servlet.
  - We will create a **HttpClient** object and **GetMethod** object for our **requestquery**. Then we will execute the method on our client object.
  - The response after executing the method will be taken as a stream so as to parse the XML elements in response message sent by the Web Service.
  - Here we used XPath to process XML response messages sent by web service. XPath Processing

XPath (XML Path Language) is a language for selecting nodes from an XML document with very advanced features.

Our response message sent by web service may look like this:
response.xml <return> <dollarToRupeesResponse>933.34</dollarToRupeesResponse>
<rupeesToEuroResponse>933.34</rupeesToEuroResponse> </return>

We will first try to get the NodeList present between <return> elements, Then we evaluate each child node to get the results.

This concludes the development section of our web based client.

Setting Up the Deployment Plan

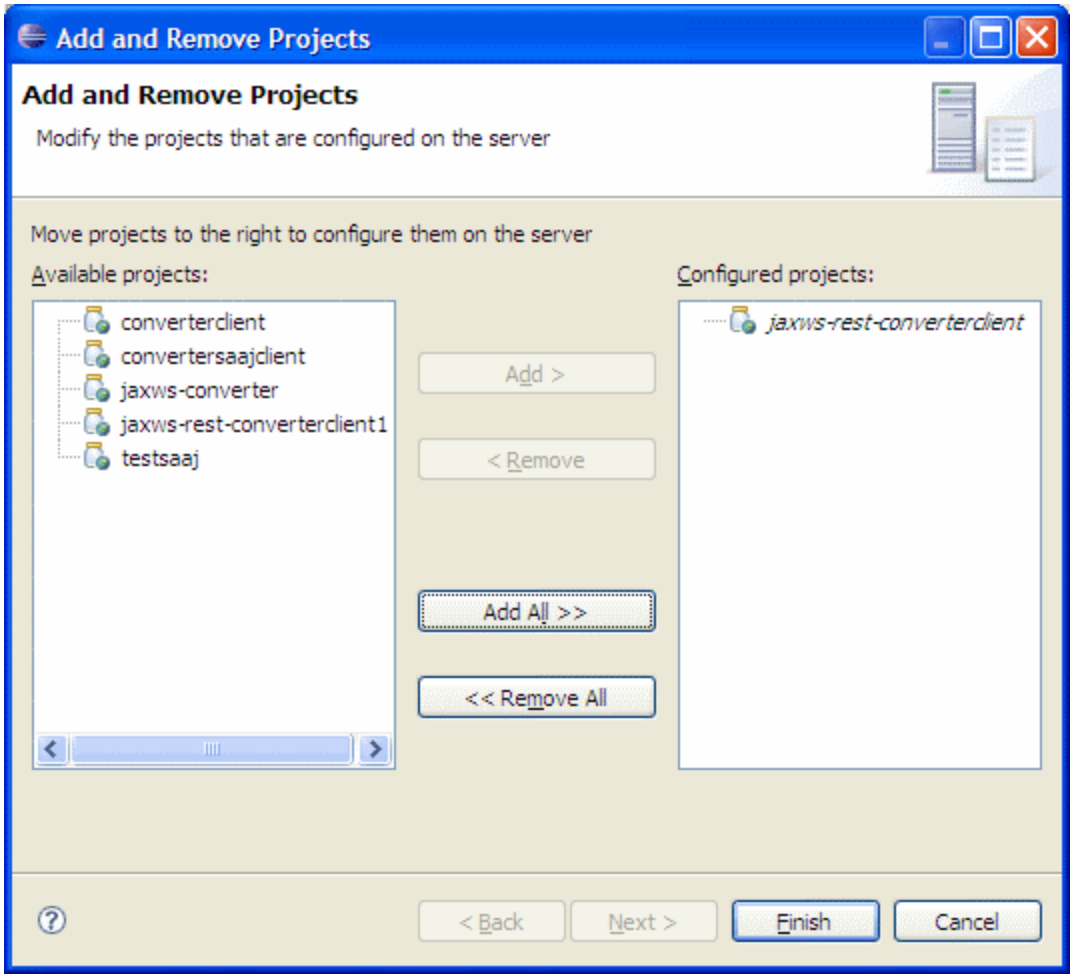
- As two external jars have been added to the project's build path, One needs to specify them as dependencies so that at runtime application can resolve the classes.
- Expand **WebContent/WEB-INF** directory and open **geronimo-web.xml**
- Add the following code to **geronimo-web.xml**

geronimo-web.xmlsolid <?xml version="1.0" encoding="UTF-8" standalone="yes"?> <ns8:web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2" xmlns:ns2="http://geronimo.apache.org/xml/ns/deployment-1.2" xmlns:ns3="http://geronimo.apache.org/xml/ns/naming-1.2" xmlns:ns4="http://openejb.apache.org/xml/ns/openejb-jar-2.2" xmlns:ns5="http://openejb.apache.org/xml/ns/pkgen-2.1" xmlns:ns6="http://geronimo.apache.org/xml/ns/j2ee/application-2.0" xmlns:ns7="http://geronimo.apache.org/xml/ns/security-2.0" xmlns:ns8="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1" xmlns:ns9="http://java.sun.com/xml/ns/persistence" xmlns:ns10="http://geronimo.apache.org/xml/ns/j2ee/application-client-2.0"> <ns2:environment> <ns2:moduleId> <ns2:groupId>default</ns2:groupId> <ns2:artifactId>jaxws-rest-converterclient</ns2:artifactId> <ns2:version>1.0</ns2:version> <ns2:type>car</ns2:type> </ns2:moduleId> <ns2:dependencies> <ns2:dependency> <ns2:groupId>commons-codec</ns2:groupId> <ns2:artifactId>commons-codec</ns2:artifactId> <ns2:version>1.3</ns2:version> <ns2:type>jar</ns2:type> </ns2:dependency> <ns2:dependency> <ns2:groupId>commons-httpclient</ns2:groupId> <ns2:artifactId>commons-httpclient</ns2:artifactId> <ns2:version>3.0.1</ns2:version> <ns2:type>jar</ns2:type> </ns2:dependency> </ns2:dependencies> </ns2:environment> <ns8:context-root>/jaxws-rest-converterclient</ns8:context-root> </ns8:web-app>

Deploying and Testing the Web Client

Deploy

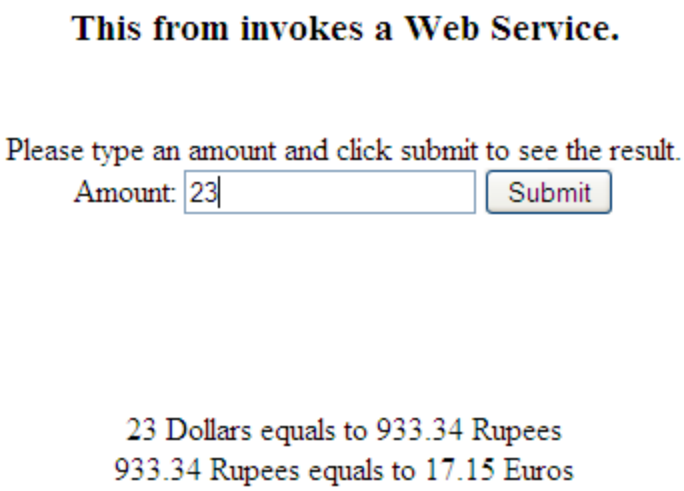
- Right click on the **Apache Geronimo** Server Runtime present in the servers view and select **Add and Remove Projects**
- Add **jaxws-rest-converterclient** to configured projects list and then click **Finish**



- Wait for some time till the server status changes to **Synchronized**

Testing

- Right click the **index.jsp** present under WebContent directory of our project and select **Run As->Run On Server**
- In the popup, check the check box **Always use this server when running the project** and then click **Finish**
- Now Eclipse will try to open the jsp in a web browser which shows you a form to enter amount in Dollars.
- Enter any amount and press **submit**, the jsp should display the result that is returned by the web service.



Test via Post
The code for testing the RESTful service via POST method hasn't been added to the ConverterHandler.
A sample post file might look like this
SampleRequest.xmlsolid <?xml version="1.0" encoding="UTF-8"?> <amount dollars="23" />

This completes the development of a simple RESTful client that consumes a RESTful Web Service that is already deployed on the server.