Developing a simple RESTful Service

{scrollbar}

This tutorial will take you through the steps required in developing, deploying and testing a RESTful Web Service in Apache Geronimo. After completing this tutorial you should be able to understand how to develop simple JAX-WS compliant RESTful web services in Apache Geronimo using Eclipse development environment.

Application Context
This application may not necessarily demonstrate why we used RESTful Web Services instead of SOAP Web Services. This tutorial only provides details about some common issues involved in developing RESTful services in Apache Geronimo.

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)
- Eclipse 3.3.1.1 (Eclipse Classic package of Europa distribution), which is platform specific
- Web Tools Platform (WTP) 2.0.1
- Data Tools Platform (DTP) 1.5.1
- Eclipse Modeling Framework (EMF) 2.3.1
- Graphical Editing Framework (GEF) 3.3.1

Details on installing eclipse are provided in the Development environment section. This tutorial takes you through the following steps:

2listpipe

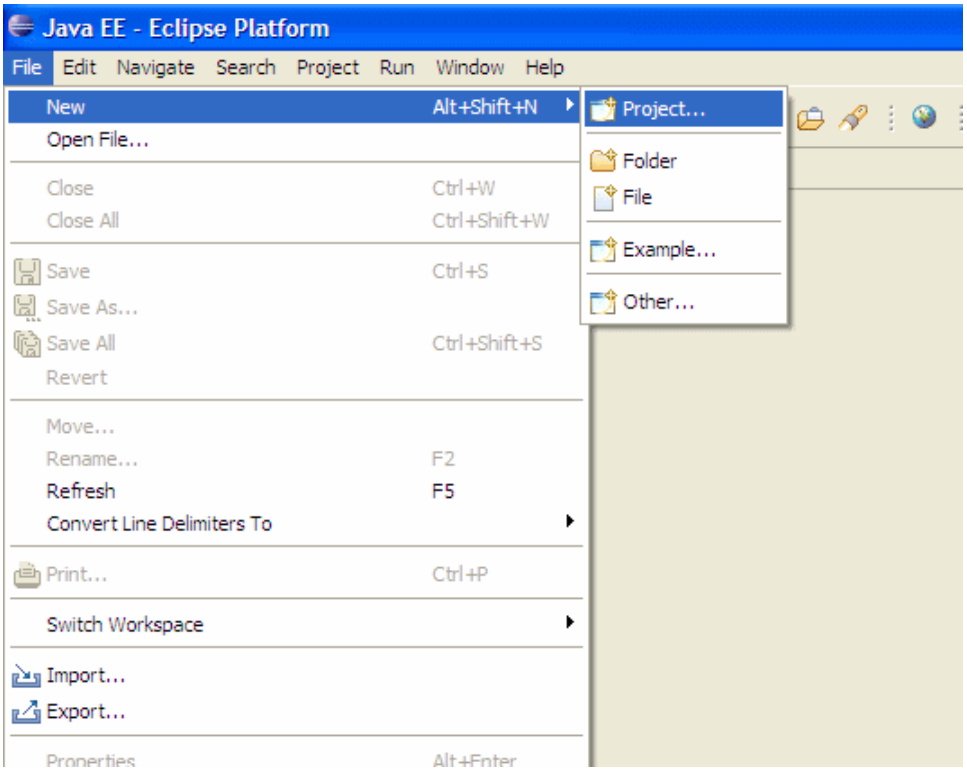
Configuring JAX-WS Engine

Firstly we need to configure our JAX-WS engine to be **Apache CXF** instead of Axis2 when using Geronimo with Tomcat. Users who are using Geronimo with Jetty don't need to worry about JAX-WS Engine as CXF is the default web services engine in Geronimo+Jetty.
For Geronimo+Tomcat users, consult this page to configure your JAX-WS engine as Apache CXF (Configure JAX-WS engine)

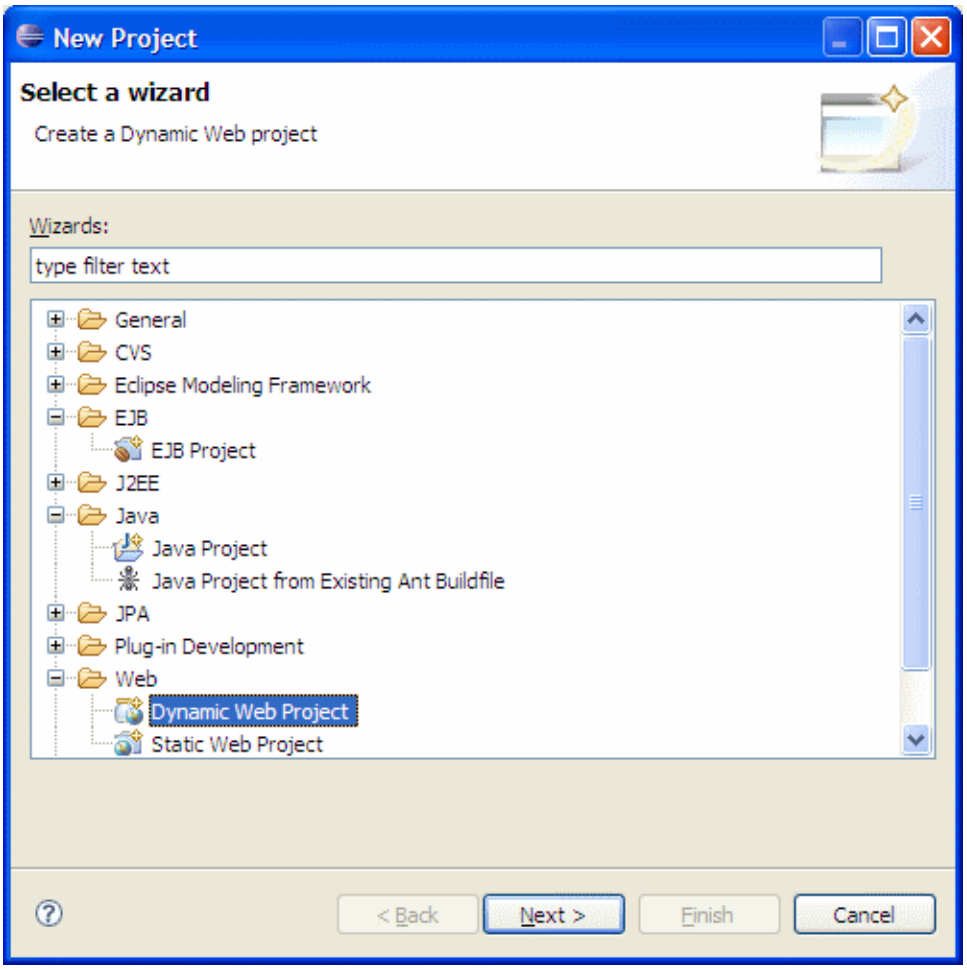
Axis2
Axis2 1.3 has/had a bug with the issue related to HTTP Content-Type header. This bug was fixed in Axis2 1.4 yet Geronimo 2.1.1 uses Axis2 1.3

Setting Up an Eclipse project

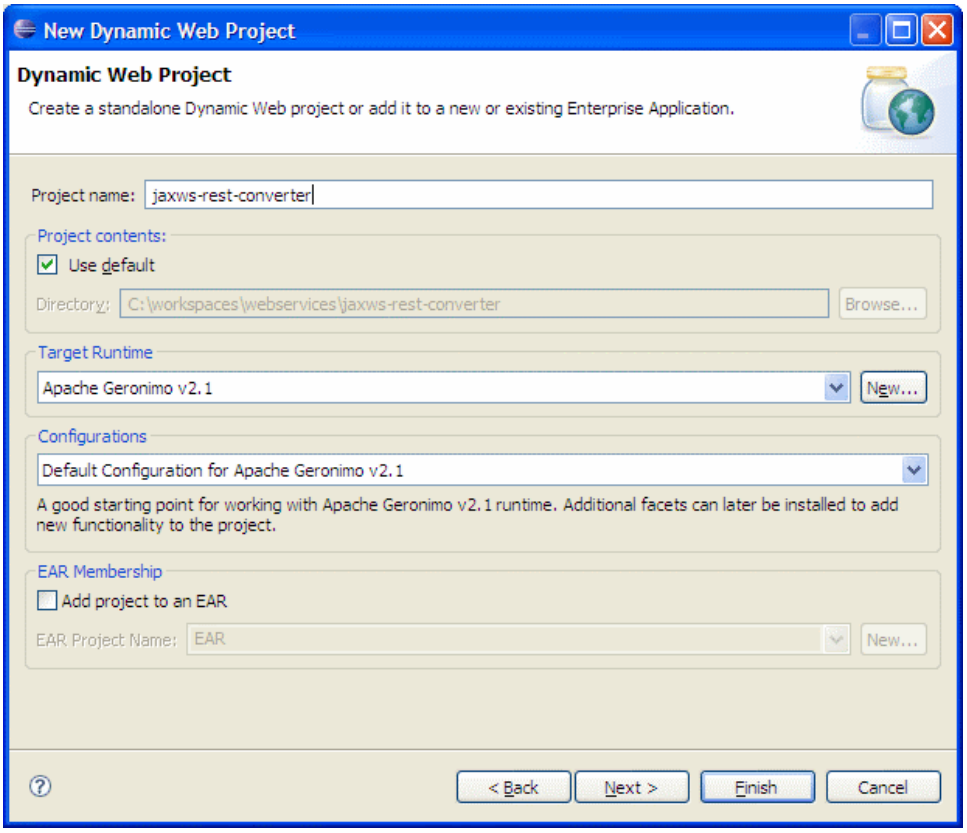
1. Create a Dynamic Web Project
    - Select **File->New->Project** (or Ctrl+N)



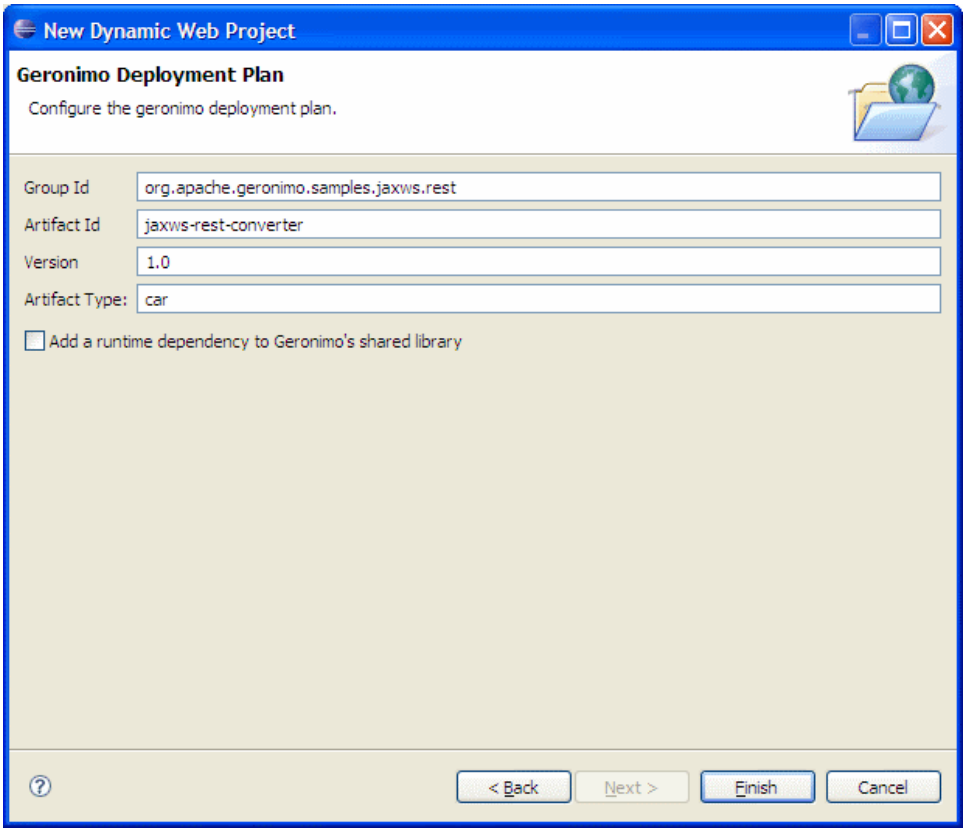
- In the popup window, select **Web->Dynamic Web Project** category (or type *dynamic* in Wizards' input field so it's left alone) and click **Next**



- Type **jaxws-rest-converter** as the **Project Name** and click **Next** twice.



- Modify the **Group Id** to **org.apache.geronimo.samples.jaxws.rest** and the **Artifact Id** to **jaxws-rest-converter**.

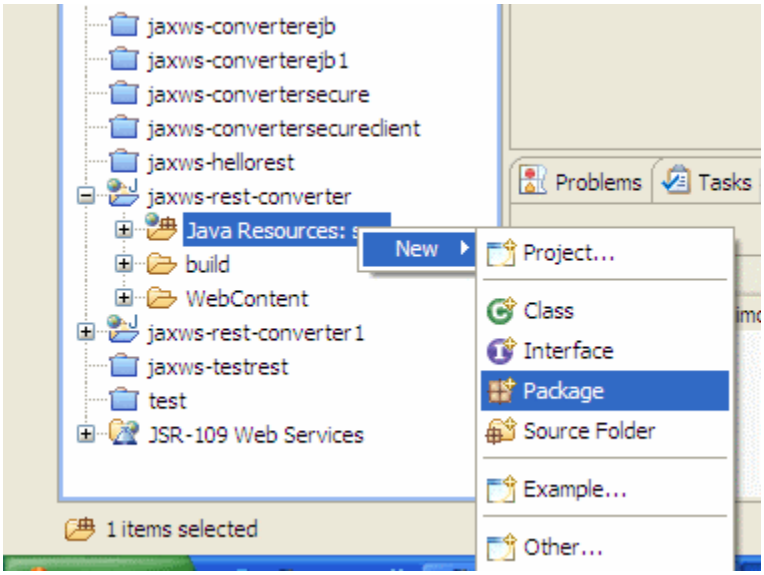


- Click **Finish**

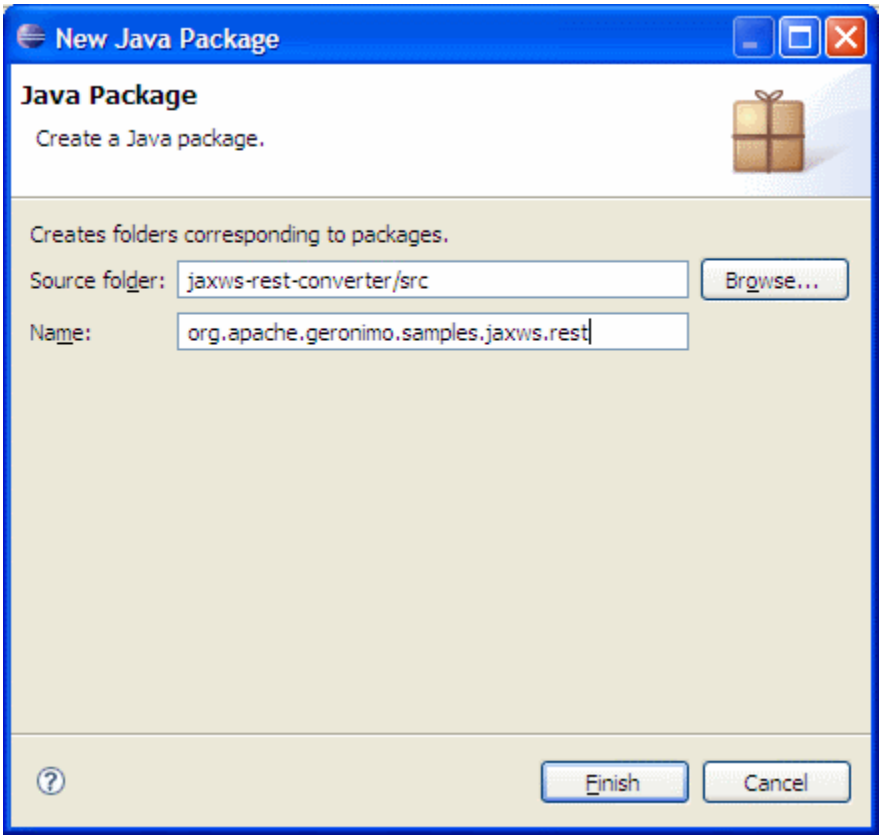
The project is now ready for further RESTful Web Services developments.

Creating the Web Services Implementation code

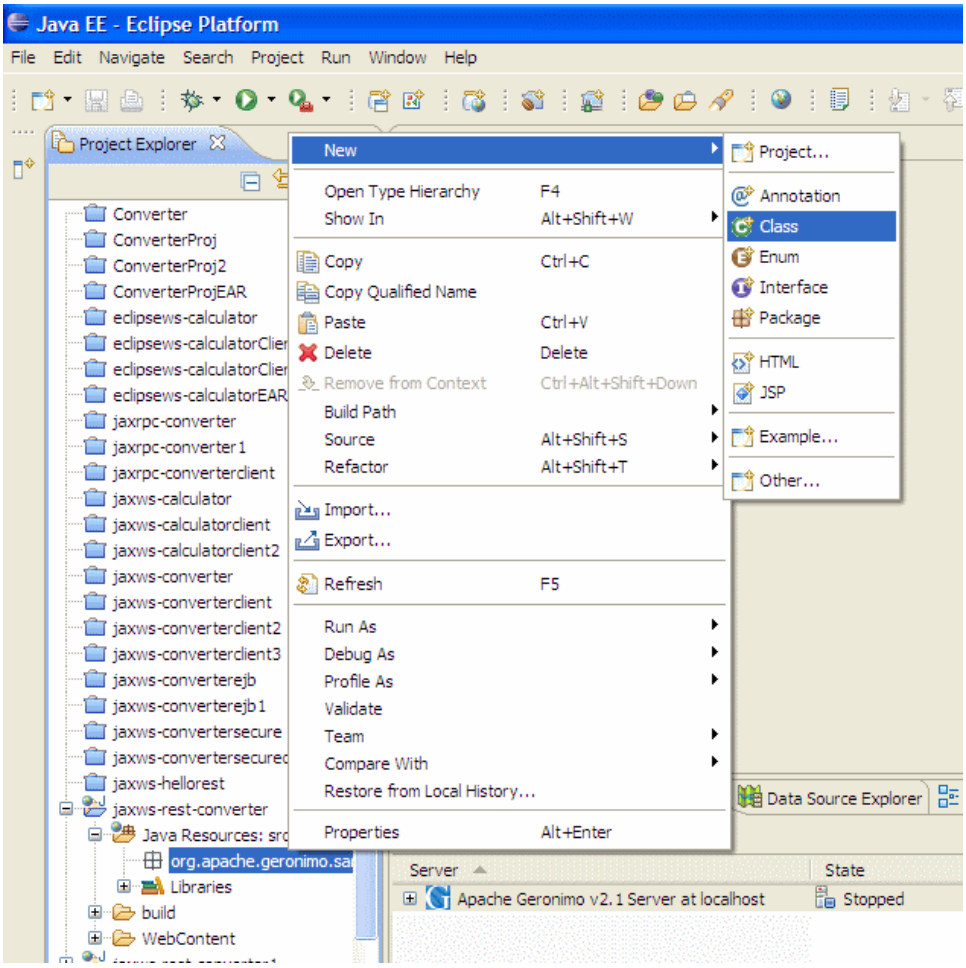
1. Right click on **JavaResources:src** and select **New->Package** (or Ctrl+N, start typing *package* and select **Package**)



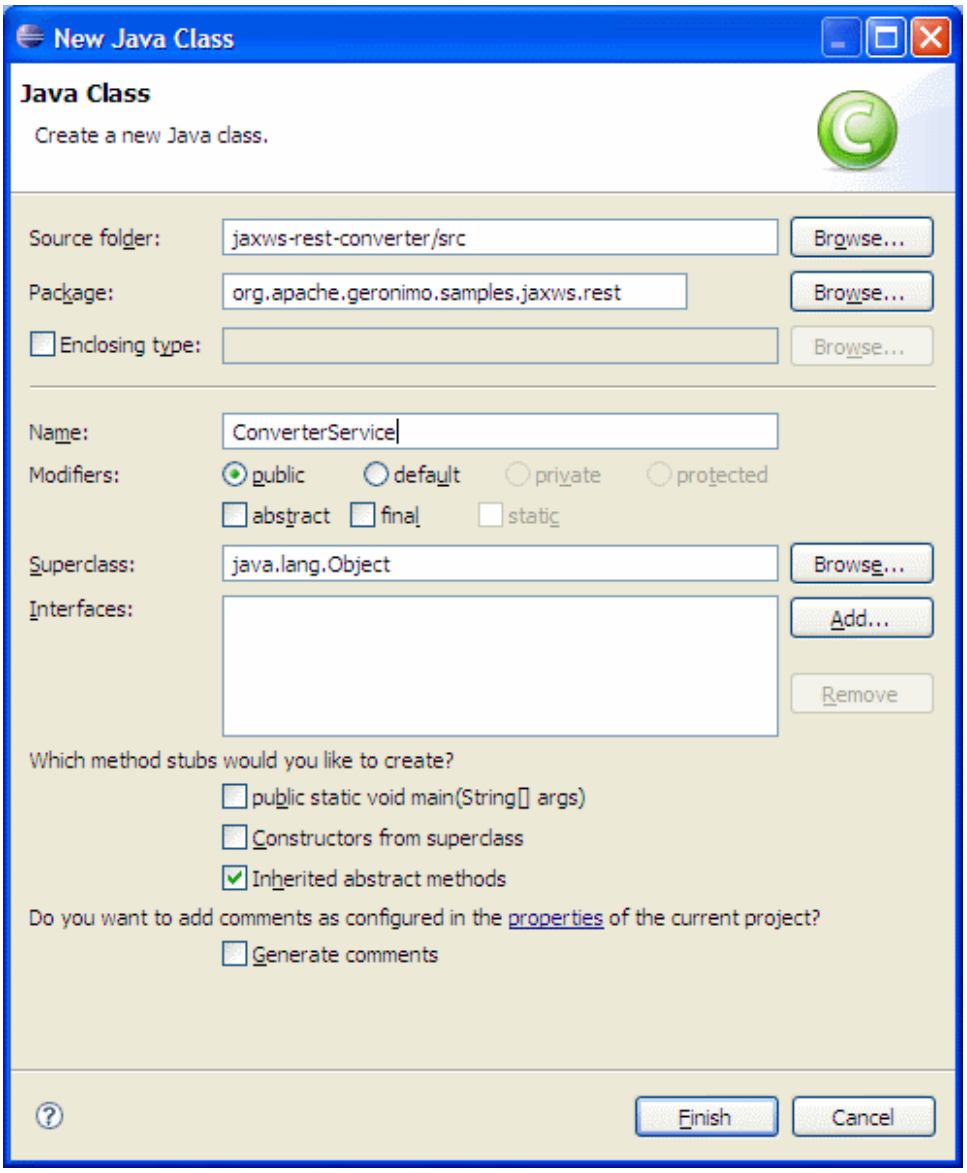
2. Name the package to **org.apache.geronimo.samples.jaxws.rest** and click **Finish**



3. Right click on the new package and select **New->Class** (or Ctrl+N, start typing *class* and select **Class**)



4. Name the class as **ConverterService** and click **Finish**



5. Add the following code to the ConverterService class ConverterService.javasolid package org.apache.geronimo.samples.jaxws.rest; import java.io.ByteArrayInputStream; import java.math.BigDecimal; import javax.annotation.Resource; import javax.servlet.ServletRequest; import javax.xml.parsers.DocumentBuilder; import javax.xml.parsers.DocumentBuilderFactory; import javax.xml.transform.Source; import javax.xml.transform.dom.DOMSource; import javax.xml.transform.stream.StreamSource; import javax.xml.ws.BindingType; import javax.xml.ws.Provider; import javax.xml.ws.WebServiceContext; import javax.xml.ws.WebServiceProvider; import javax.xml.ws.handler.MessageContext; import javax.xml.ws.http.HTTPBinding; import javax.xml.ws.http.HTTPException; import org.w3c.dom.Node; import org.w3c.dom.NodeList; import org.xml.sax.InputSource; @WebServiceProvider @BindingType(value = HTTPBinding.HTTP_BINDING) public class ConverterService implements Provider<Source> { @Resource protected WebServiceContext wsContext; private BigDecimal rupeeRate = new BigDecimal("40.58"); private BigDecimal euroRate = new BigDecimal("0.018368"); public Source invoke(Source source) { try { String amount = null; if (source == null) { System.out.println("Getting input from query string"); MessageContext mc = wsContext.getMessageContext(); String query = (String) mc.get(MessageContext.QUERY_STRING); System.out.println("Query String = " + query); ServletRequest req = (ServletRequest) mc.get(MessageContext.SERVLET_REQUEST); amount = req.getParameter("amount"); } else { System.out.println("Getting input from input message"); Node n = null; if (source instanceof DOMSource) { n = ((DOMSource) source).getNode(); } else if (source instanceof StreamSource) { StreamSource streamSource = (StreamSource) source; DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance(); DocumentBuilder db = dbf.newDocumentBuilder(); InputSource inputSource = null; if (streamSource.getInputStream() != null) { inputSource = new InputSource(streamSource.getInputStream()); } else if (streamSource.getReader() != null) { inputSource = new InputSource(streamSource.getReader()); } n = db.parse(inputSource); } else { throw new RuntimeException("Unsupported source: " + source); } NodeList children = n.getChildNodes(); for (int i = 0; i < children.getLength(); i++) { Node child = children.item(i); if (child.getNodeName().equals("dollars")) { amount = child.getAttributes().getNamedItem("amount").getNodeValue(); break; } } } BigDecimal dollars = new BigDecimal(amount); BigDecimal rupees = dollarToRupees(dollars); BigDecimal euros = rupeesToEuro(rupees); return createResultSource(rupees, euros); } catch (Exception e) { e.printStackTrace(); throw new HTTPException(500); } } public BigDecimal dollarToRupees(BigDecimal dollars) { BigDecimal result = dollars.multiply(rupeeRate); return result.setScale(2, BigDecimal.ROUND_UP); } public BigDecimal rupeesToEuro(BigDecimal rupees) { BigDecimal result = rupees.multiply(euroRate); return result.setScale(2, BigDecimal.ROUND_UP); } private Source createResultSource(BigDecimal rupees, BigDecimal euros) { String body = "<ns:return xmlns:ns=\"http://rest.jaxws.samples.geronimo.apache.org\">" + "<ns:dollarToRupeesResponse>" + rupees + "</ns:dollarToRupeesResponse><ns:rupeesToEurosResponse>" + euros + "</ns:rupeesToEurosResponse></ns:return>"; Source source = new StreamSource(new ByteArrayInputStream(body.getBytes())); return source; } }

This completes the development of Web Services implementation code.
Now let us walk through the code that we just developed.

- **@WebServiceProvider** - This annotation specifies that our web service works at XML Level of the message by extending a generic **Provider** Interface.
- **Provider** - Here our service extended the **Provider** by passing the generic argument as **Source**
- **Source** - This contains the information needed to act as source input (XML Source). We use this argument to send our request in XML File.
- Our ConverterService can handle two types of requests i.e **GET** and **POST**.
    - **GET** - Here we send the argument to the service in URL in which case the input source argument will be null for the Provider.
    - **POST** - We send our request in an XML file by posting it at the URL where web service is located. Here XML file is passed as argument to Provider.
- **CreateResultSource** - Here we need to return the response in a properly formatted XML message from the Provider with the results.

Setting Up the Deployment Descriptor and Deployment Plan

There isn't much to do in this part for deploying RESTful Web Services except that we need to expose our ConverterService as servlet and CXF automatically generates the required files.

- Expand **WEB-INF** directory and add the following code to the deployment descriptor (web.xml) web.xmlsolid <?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" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5"> <display-name>jaxws-rest-converter</display-name> <servlet> <servlet-name>ConverterService</servlet-name> <servlet-class> org.apache.geronimo.samples.jaxws.rest.ConverterService </servlet-class> <load-on-startup>0</load-on-startup> < /servlet> <servlet-mapping> <servlet-name>ConverterService</servlet-name> <url-pattern>/converter</url-pattern> </servlet-mapping> </web-app>

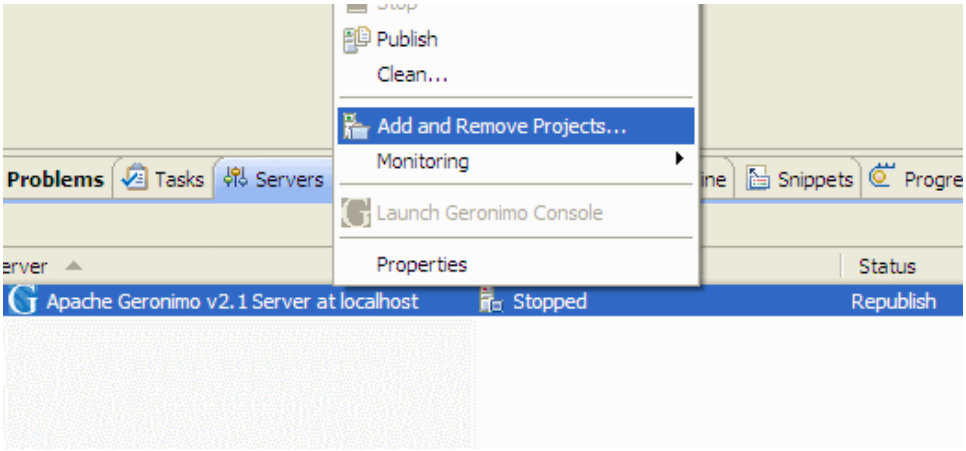
This completes the setting up of Deployment Descriptor and Deployment Plan.

Deploy and Test the Web Service

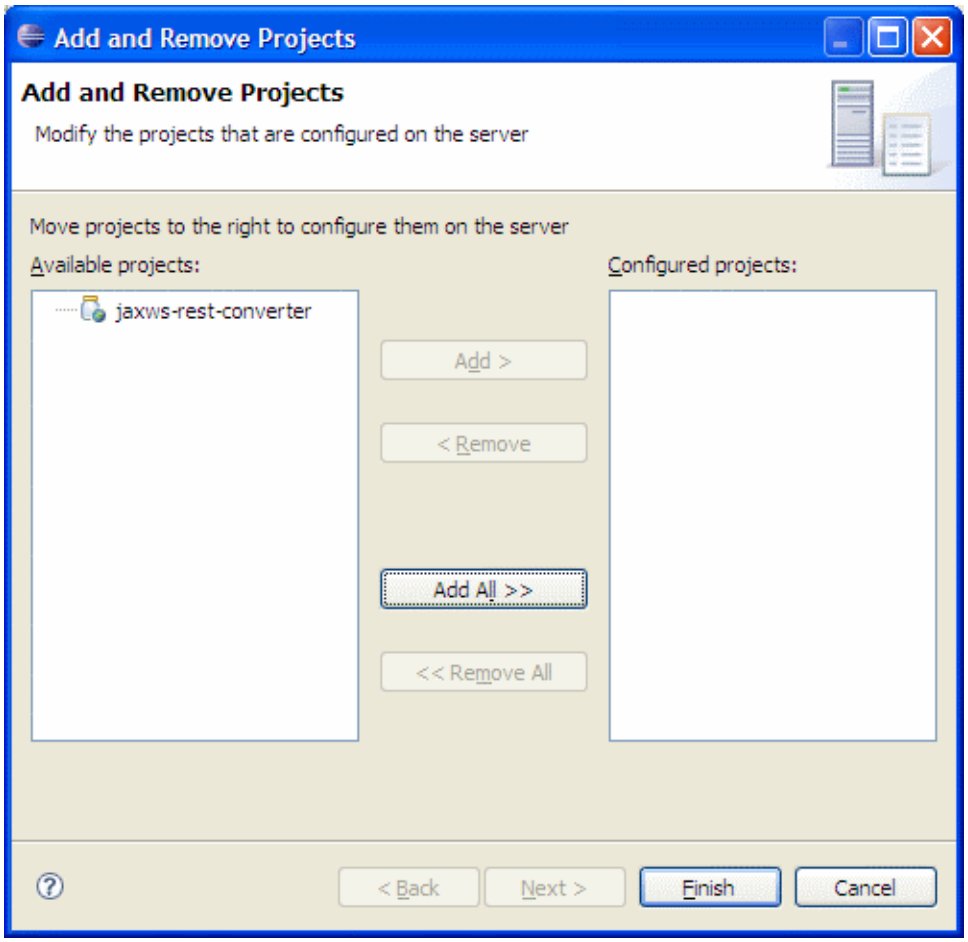
Now, we will look into the steps involved in deploying and testing our web service without any clients.

Deploy

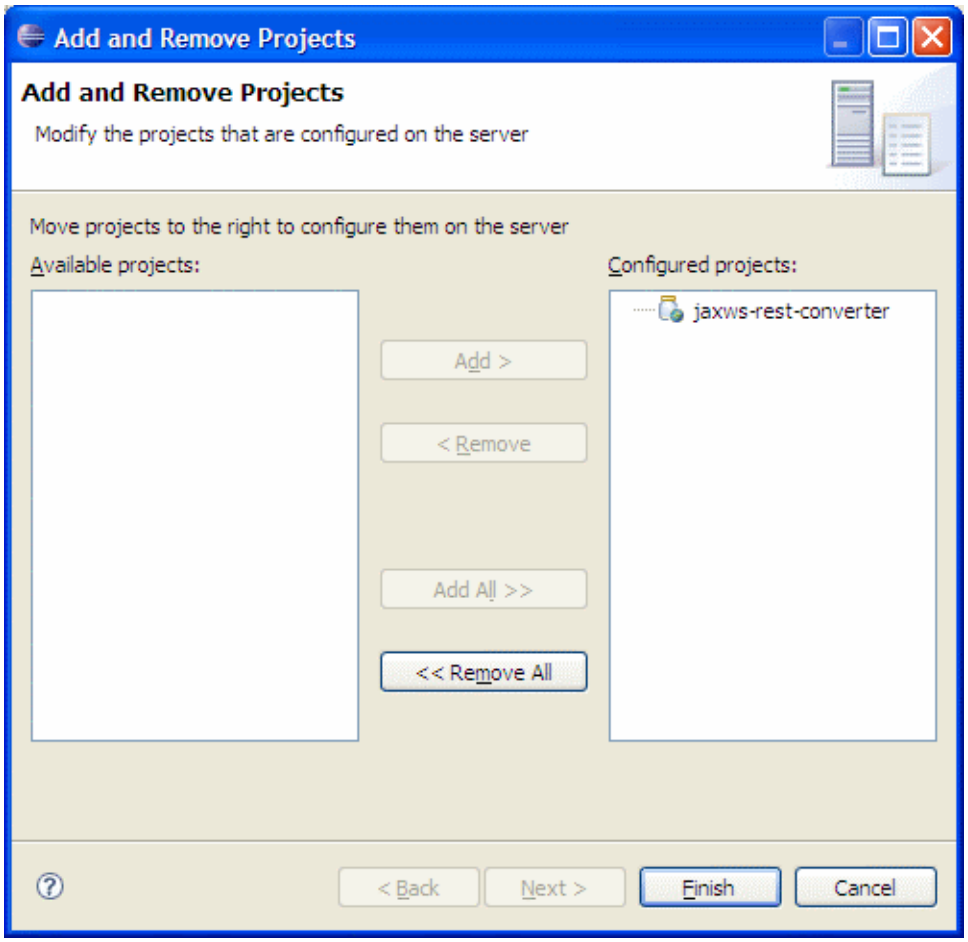
- Right click on the **Apache Geronimo** Server Runtime available in the Servers view and select **Add or Remove Projects**



- In the popup dialog, select the **jaxws-rest-converter** project and click **Add**



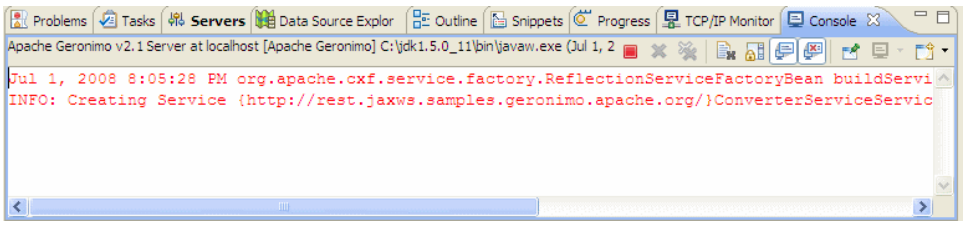
- Make sure that **jaxws-rest-converter** is in the **Configured projects** panel and click **Finish**



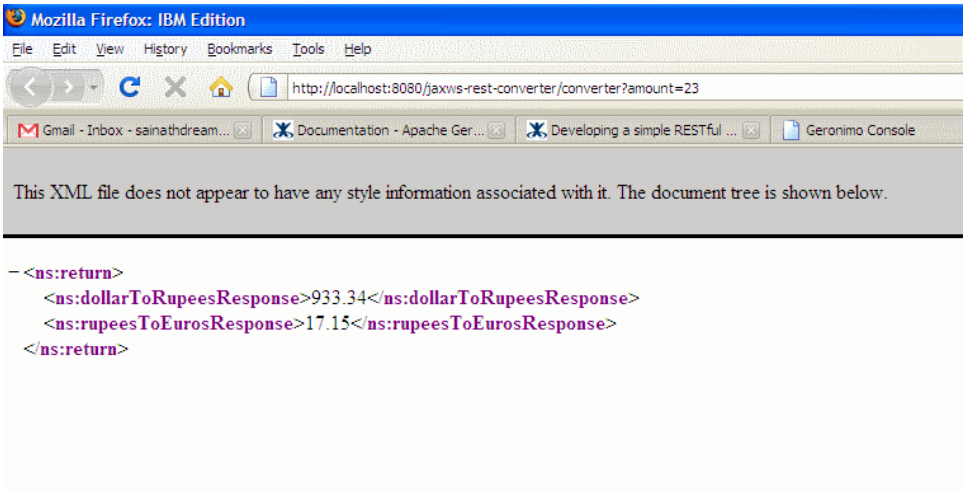
- Wait a little bit till the server status is changed to synchronized (start Geronimo if it has not been started before).

Test

- You should see CXF reporting that it's created Service from our Service implementation.



- Finally we can also test the **GET** method of accessing our REST Service.
  - Go to the following URL http://localhost:8080/jaxws-rest-converter/converter?amount=23
  - You should see the following screen reporting the results of Converter



This is just a tip of iceberg in RESTful Web Services. There's still a lot to explore in RESTful services.