SAAJ Messaging Web Services

{scrollbar}

This article takes you through the some of basic concepts involved in **SAAJ Messaging**. The goal of this tutorial is to give a brief overview about the terminology involved with SOAP (or SAAJ) messages.

We will also develop a basic SAAJ client that accesses a deployed web service by sending SOAP messages. By using SAAJ we will working at **XML level of messages** which means that we need to create **SOAP request** messages and parse the **SOAP response** messages for results.

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

The steps that we follow in due course of the tutorial are:

2listpipe

What is SAAJ?

SAAJ stands for **SOAP with Attachments API for Java**. SAAJ messages follow the SOAP standards, which prescribe the format for messages. With **SAAJ API** one can create XML messages that conform to SOAP 1.1 or 1.2 specification by simple making Java API calls.

SAAJ Messages and Connections

Messages

The two main types of SOAP messages are those that have attachments and those that do not.Messages sent using the SAAJ API are called **request-response** messages

Outline of SOAP message
The following outline shows the very high-level structure of a SOAP message.

- SOAP message
    - SOAP part
        - SOAP envelope
            - SOAP header (optional)
            - SOAP body
    - Attachment Part
        - MIME Headers
        - Content

Connections

All SOAP messages are sent and received over a connection. With the SAAJ API, the connection is represented by a **SOAPConnection** object, which goes from the sender directly to its destination. They are sent over a SOAPConnection object with the **call** method, which sends a message (a request) and then blocks until it receives the reply (a response).
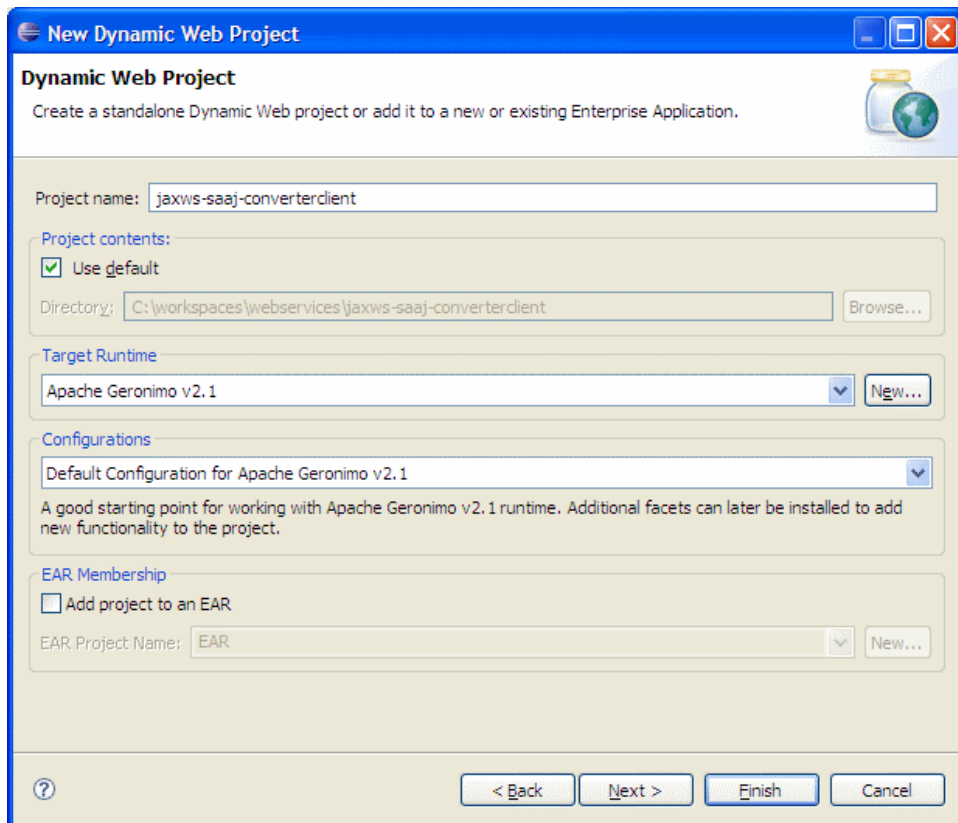
Develpoing a SAAJ Client

Web Service Deployed
The SAAJ client that we are going to develop is targeted towards the web service that we deployed in the Developing a JAX-WS POJO Web Service. Although this model will work any web service if we change the request message according to the **WSDL file** of deployed service.

Creating a Dynamic Web Project

- 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**
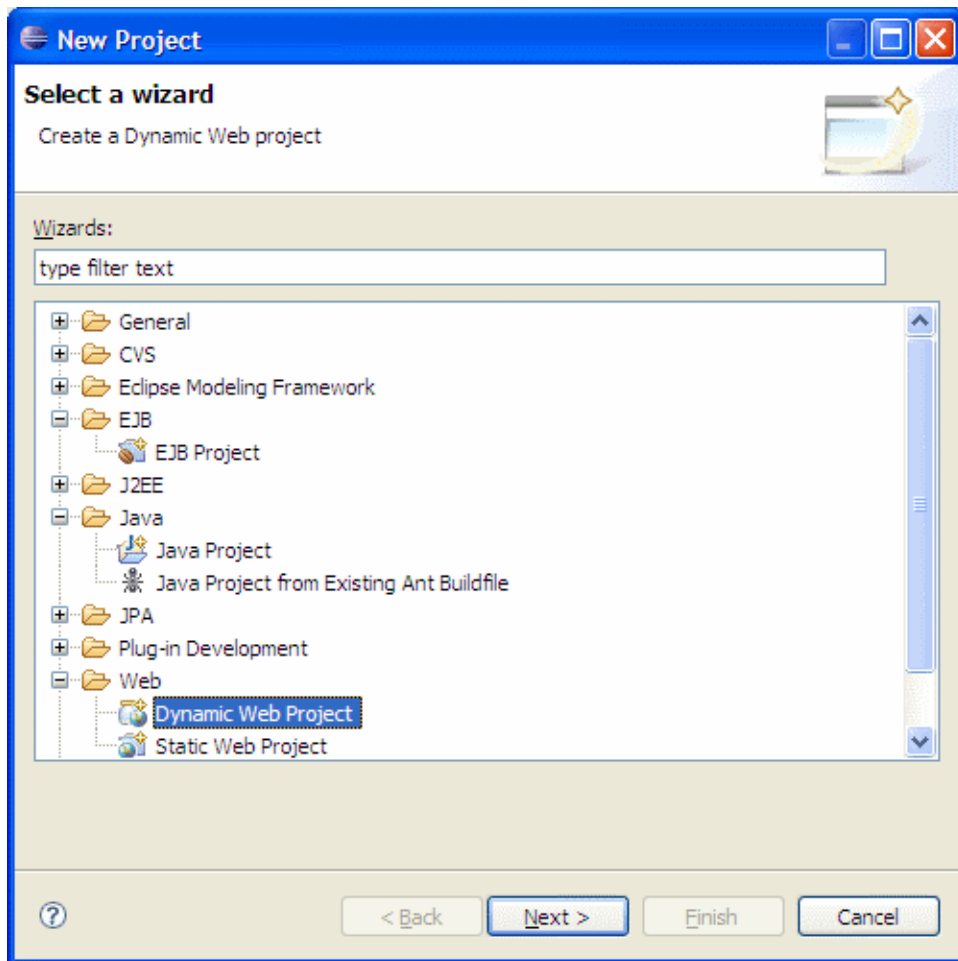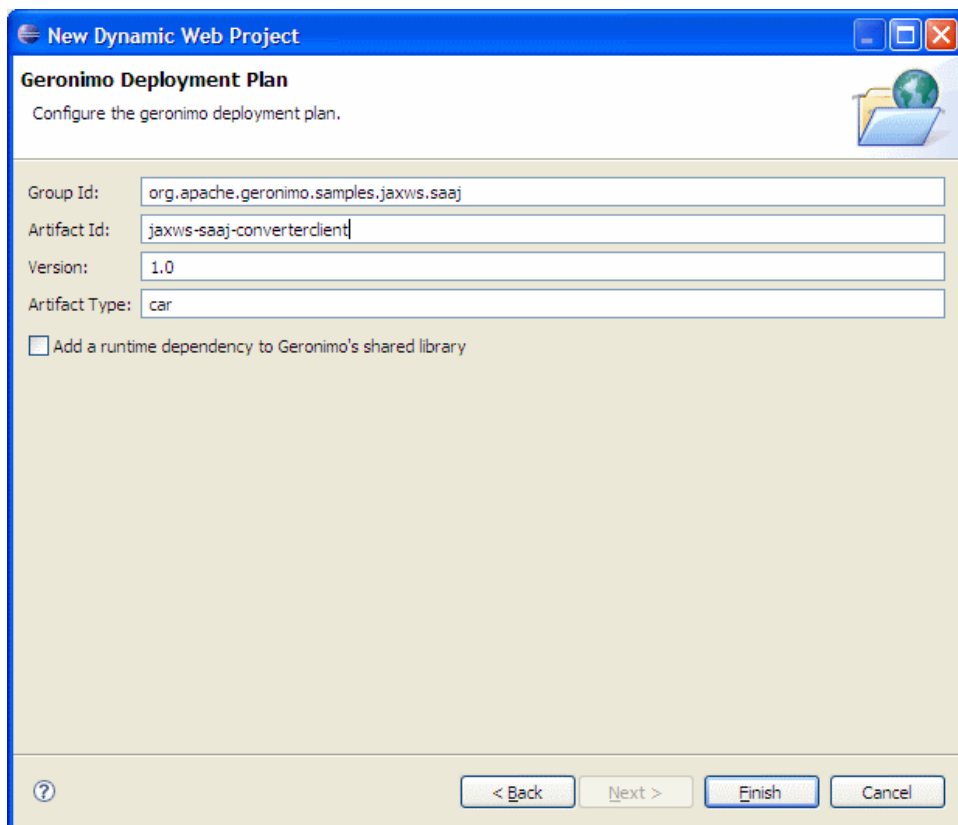
- Type **jaxws-saaj-converterclient** as the **Project Name** and click **Next** twice.

○ Modify the **Group Id** to **org.apache.geronimo.samples.jaxws.saaj** and the **Artifact Id** to **jaxws-saaj-converterclient**.

○ Click **Finish**

# Adding code to send and receive SOAP messages

- Right Click the **jaxws-saaj-converterclient**, and Select **New->JSP**

- Name the jsp as **index.jsp** and click **Finish**
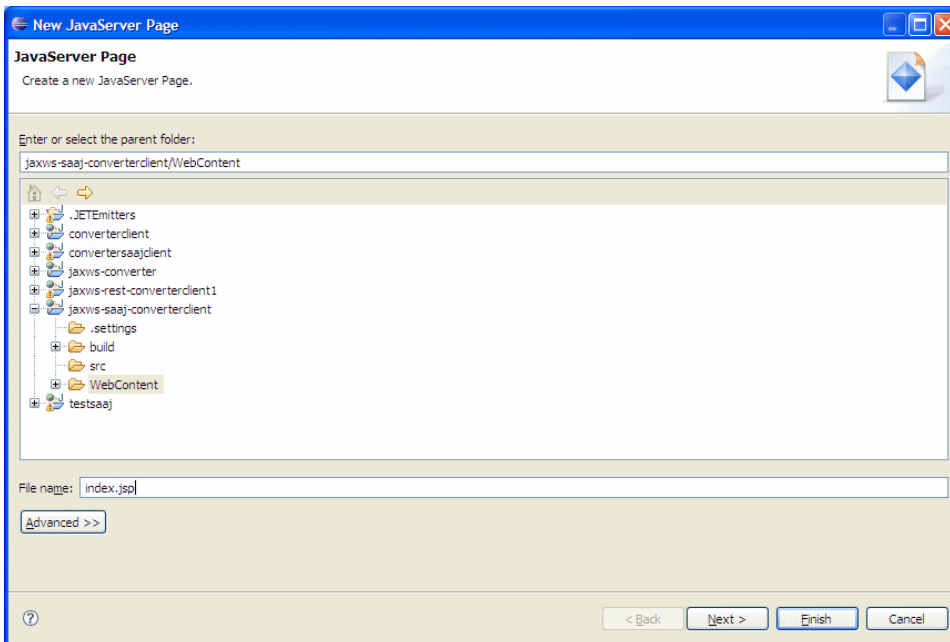
- Add the following code to the **index.jsp**

index.jspsolid <%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%> <html> <head> <title>Converter</title> <meta content="text/html; CHARSET=iso-8859-1" http-equiv="Content-Type"> </head> <body> <center> <h3>This from invokes a Web Service.</h3>
 Please type an amount and click submit to see the result.
 <form action="index.jsp">Amount: <input type="text" name=" amount"> <input type="submit" value="Submit"></form>
 <jsp:include page="ConverterHandler"></jsp:include></center> </body> </html>

- Right click again and add a **Servlet** named **ConverterHandler**
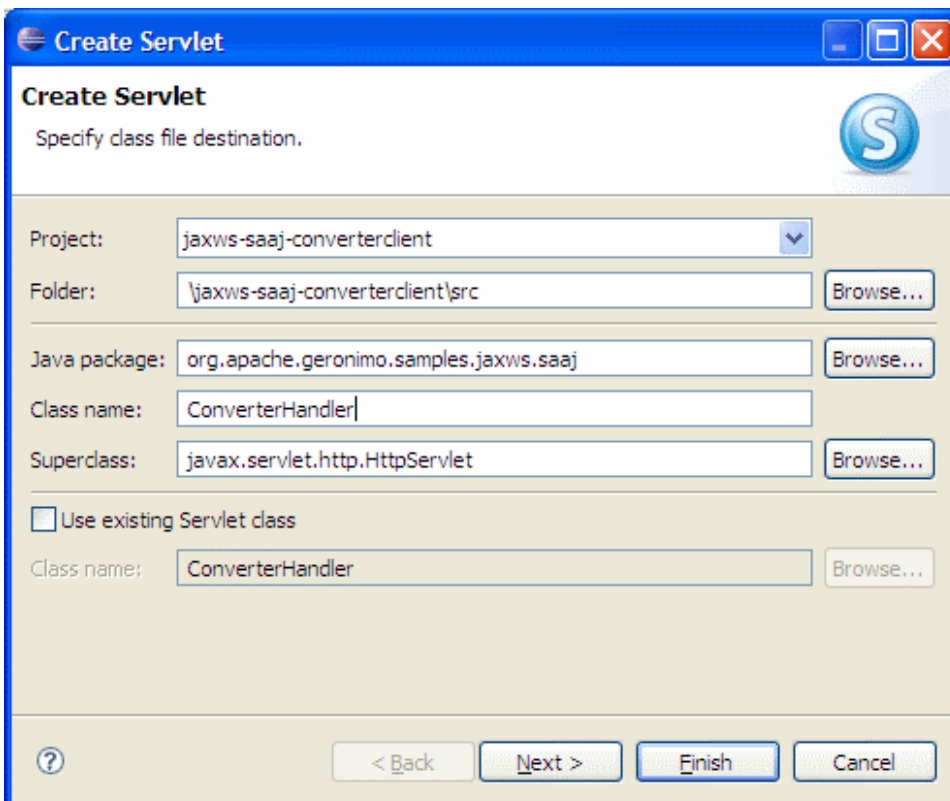
- Add the following code to **ConverterHandler.java**

ConverterHandler.javasolid package abc; import java.io.IOException; import java.io.PrintWriter; import java.util.Iterator; import javax.servlet. ServletException; import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse; import javax.xml.soap.MessageFactory; import javax.xml.soap.SOAPBody; import javax.xml.soap.SOAPConnection; import javax.xml.soap.SOAPConnectionFactory; import javax.xml.soap. SOAPElement; import javax.xml.soap.SOAPEnvelope; import javax.xml.soap.SOAPException; import javax.xml.soap.SOAPMessage; import javax.xml. soap.SOAPPart; import org.w3c.dom.Node; 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 { String dollars = request.getParameter("amount"); if (dollars != null && dollars.trim().length() > 0) { String rupees = null, euros = null; try { rupees = returnResult(createSOAPMessage("dollarToRupees", dollars)); euros = returnResult(createSOAPMessage ("rupeesToEuro", rupees)); } catch (SOAPException e) { // TODO Auto-generated catch block e.printStackTrace(); } PrintWriter out = response.getWriter(); out.println("

"); out.println(dollars + " Dollars equals to " + rupees + " Rupees"); out.println("<br>"); out.println(rupees + " Rupees equals to " + euros + " Euros"); } } protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { doGet (request, response); } public SOAPMessage createSOAPMessage(String operation, String arg) throws SOAPException { String urn = "http://jaxws.samples. geronimo.apache.org"; MessageFactory messageFactory; SOAPMessage message = null; messageFactory = MessageFactory.newInstance(); message = messageFactory.createMessage(); SOAPPart soapPart = message.getSOAPPart(); SOAPEnvelope envelope = soapPart.getEnvelope(); SOAPBody body = envelope.getBody(); SOAPElement bodyElement = body.addChildElement(envelope.createName(operation, "ns1", "urn:" + urn)); bodyElement. addChildElement("arg0").addTextNode(arg); message.saveChanges(); return message; } public String returnResult(SOAPMessage message) throws SOAPException { String destination = "http://localhost:8080/jaxws-converter/converter"; SOAPConnectionFactory soapConnFactory = SOAPConnectionFactory .newInstance(); SOAPConnection connection = soapConnFactory.createConnection(); SOAPMessage reply = connection.call (message, destination); SOAPPart soapPart = reply.getSOAPPart(); SOAPEnvelope envelope = soapPart.getEnvelope(); SOAPBody body = envelope. getBody(); Iterator iter = body.getChildElements(); Node resultOuter = ((Node) iter.next()).getFirstChild(); Node result = resultOuter.getFirstChild(); connection.close(); return result.getNodeValue(); } }

- Let us have a brief look at the code that we added in **ConverterHandler.java**
  - **createSOAPMessage()** - Here we will create a new SOAP message from **MessageFactory** instance and set the SOAP body of message according to the request and format needed by WSDL file.
    SOAP request message that is returned by createSOAPMessage for operation ("dollarToRupees") and argument ("23") looks like this: S OAP Request Message <?xml version="1.0" encoding="UTF-8"?> <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org /soap/envelope/"> <soapenv:Body> <ns1:dollarToRupees xmlns:ns1="urn:http://jaxws.samples.geronimo.apache.org"> <arg0>23< /arg0> </ns1:dollarToRupees> </soapenv:Body> </soapenv:Envelope>
  - **returnResult()** - This function processes the SOAP response message sent by the Web service and returns the result. This function uses **call** method over a **SOAP Connection** to send the request and receive the response.
    SOAP response message that is returned by Web Service for the above SOAP Request message looks like this: SOAP Response Message <?xml version="1.0" encoding="UTF-8"?> <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"> <soapenv:Header/> <soapenv:Body> <dlwmin:dollarToRupeesResponse xmlns:dlwmin="http://jaxws.samples.geronimo.apache.org"> <axis2ns1:return>933.34</axis2ns1:return> </dlwmin:dollarToRupeesResponse> </soapenv:Body> </soapenv:Envelope>
  - Here the SOAP Response is parsed by using the functions present in SAAJ API. Also observe that **call** is a blocking call which means that it will continue waiting until it receives a response.

This concludes the development section of our web based client.

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-saaj-converterclient** to configured projects list and then click **Finish**
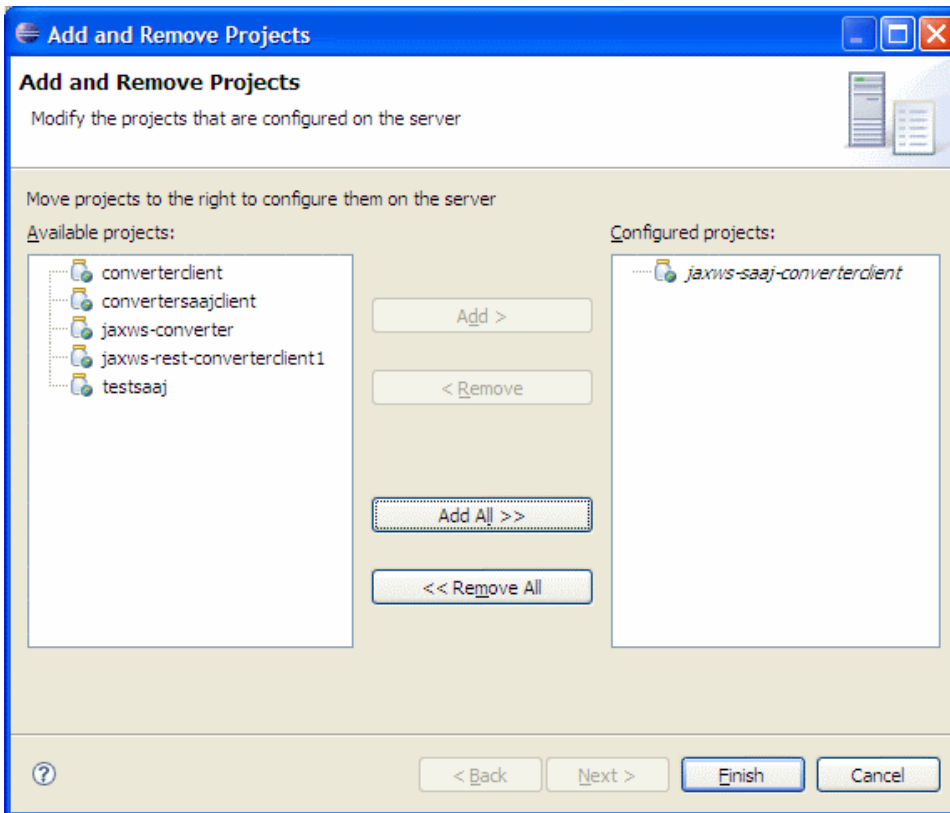
- Wait for some time till the server status changes to **Synchronized**

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.

This completes our development and deployment of a basic SAAJ client that works at XML level by sending SOAP request messages to the deployed Web Service.