Development Environment

JDK 1.8

Eclipse - Oxygen version

<https://www.eclipse.org/downloads/packages/release/oxygen/3a>

Axis2 - v 1.64

<https://archive.apache.org/dist/axis/axis2/java/core/1.6.4/>

axis2-1.6.4-bin.zip core

axis2-eclipse-codegen-plugin-1.6.4.zip eclipse codegen plugin

Tomcat - v9

<https://tomcat.apache.org/download-90.cgi>

Smartbear - SoapUI

<https://www.soapui.org/tools/soapui/>

If you use the toolset I uploaded Eclipse will expect to find Axis2, JDK1.8, and Tomcat in the following locations:

Axis2 - C:\opt\axis2\axis2-1.6.4

JDK - C:\opt\Java\jdk1.8.0\_192

Tomcat - C:\opt\apache-tomcat-9.0.37

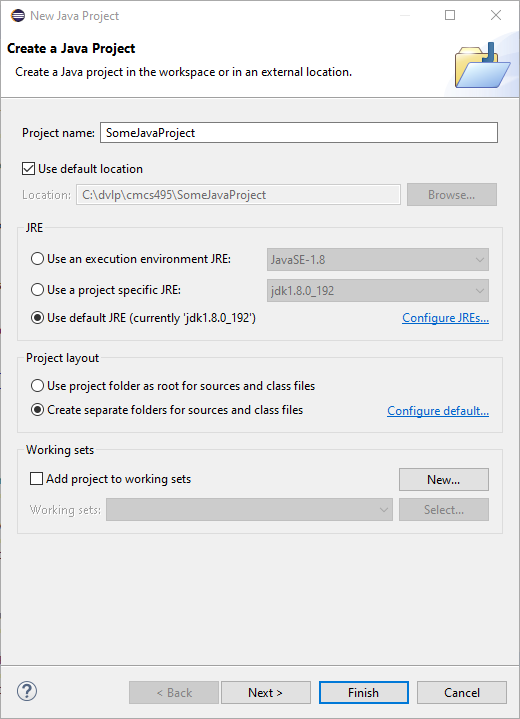
In Eclipse you need to specify the JDK, Tomcat, and Axis2 installations in the Preferences option under the Windows option on the top level menu.

* -The JDK is specified under the Java->Installed JREs option.
* -The Tomcat instance is specified under the Server->Runtime Environment option.
* -The Axis2 instance is specified under the Web Services->Axis2 Preferences option.

**Service Generation**

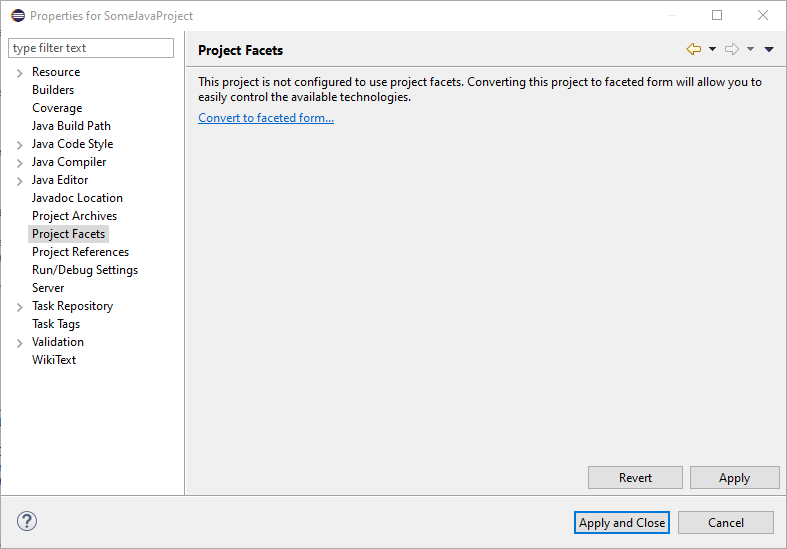
This service was generated in a 'top down' fashion meaning we started with a WSDL file and created Java classes for marshalling and unmarshalling SOAP requests/responses.

Both of the eclipse projects were created as basic Java Projects:

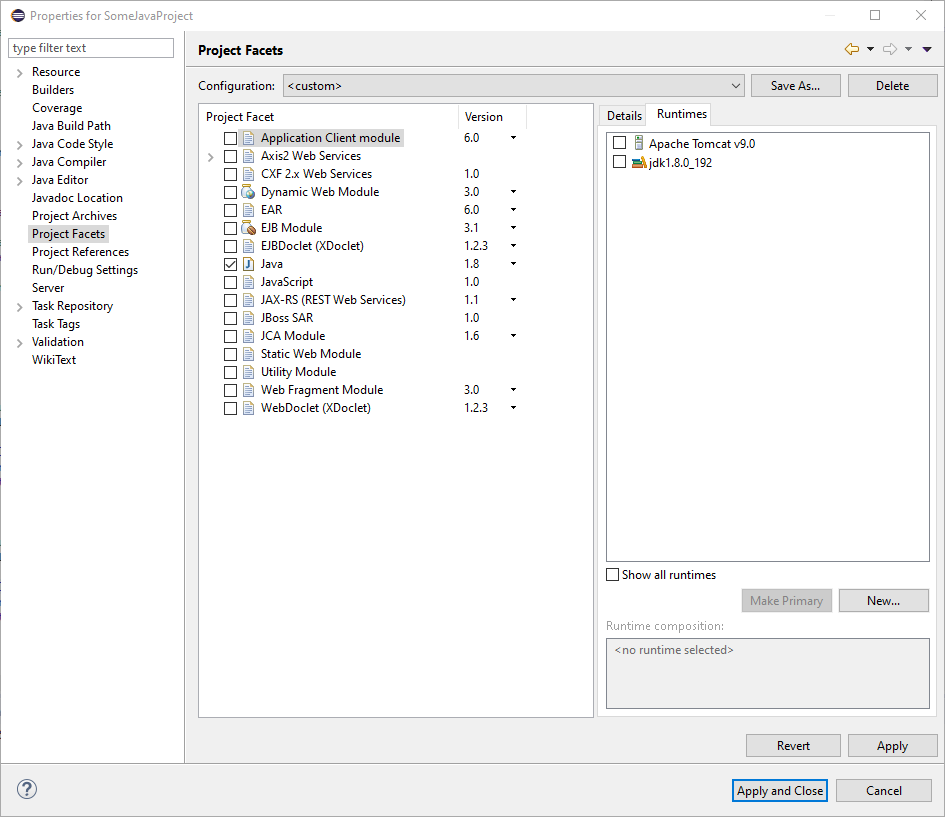


Right click on newly created Java project and select 'Properties' which is the last item in the pop-up. The properties dialog should appear for the project.

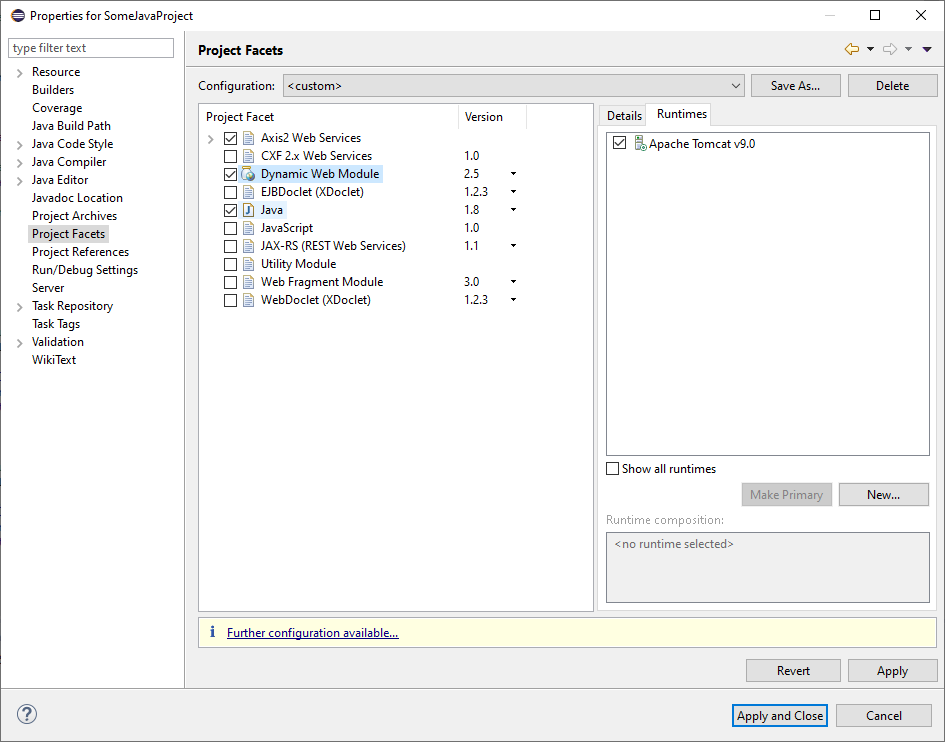
Select Project Facets option on the left and then select the 'Convert to faceted form...' text in the center of the dialog.



Under the Runtimes tab on the right select Apache Tomcat v9.0. If you have a jdk defined there ignore it.



In the center of the dialog under 'Project Facet', select Axis2 Web Services, Dynamic Web Module, and Java 1.8. For the Dynamic Web Module you will need to select version 2.5.

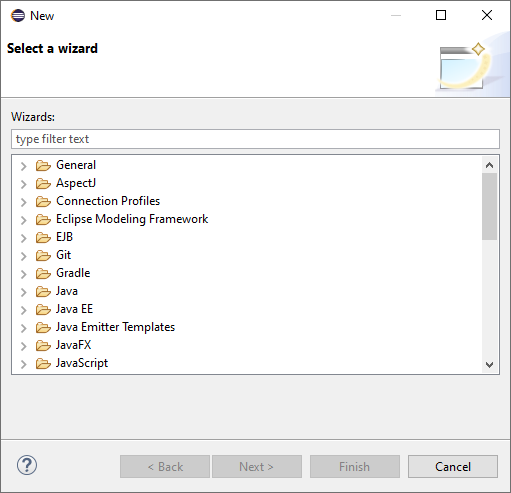


Select 'Apply and Close'

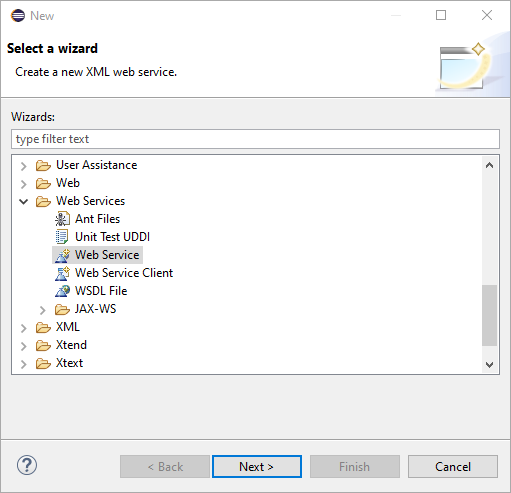
At this point your newly created project will be a Java project with Web characteristics that include Axis2.

Finally you will use the WSDL file to generate the classes utilized to create SOAP request/response instances and the stub/skeleton classes for marshalling/demarshalling SOAP req/rsp and calling the service.

In the project just created which is going to be the SOAP service, right click on it and select 'New' and at the very bottom of the pop up select 'other' which should give you the 'Select a Wizard' dialog.



Scroll down to the 'Web Services' option and select 'Web Service' within.

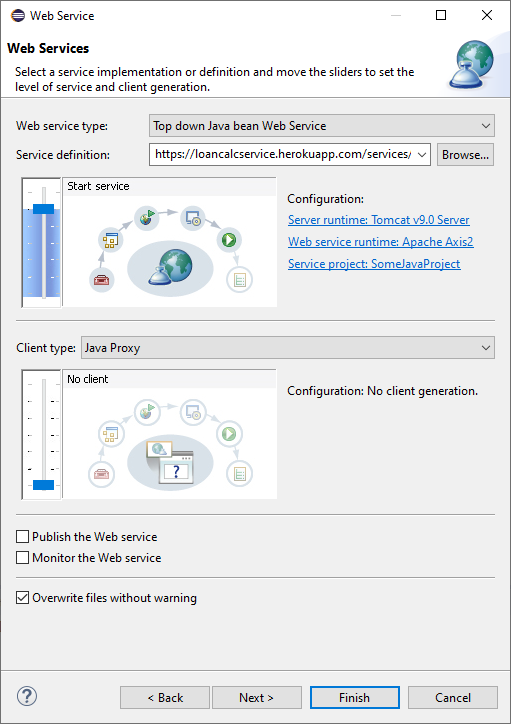


In the Web Service dialog set options as indicated.

(Service definition can be a file or a url)

<https://loancalcservice.herokuapp.com/services/LoanCalcService?wsdl> or

LoanCalcService.wsdl in the root of the LoanCalcService project.



Select 'Finish' and you should see this dialog.

Service Name, Port Name and the Namespaces below are pulled from the WSDL.

We are using the basic Databinding for Axis. Could use Xmlbeans as well but this will have less moving parts.

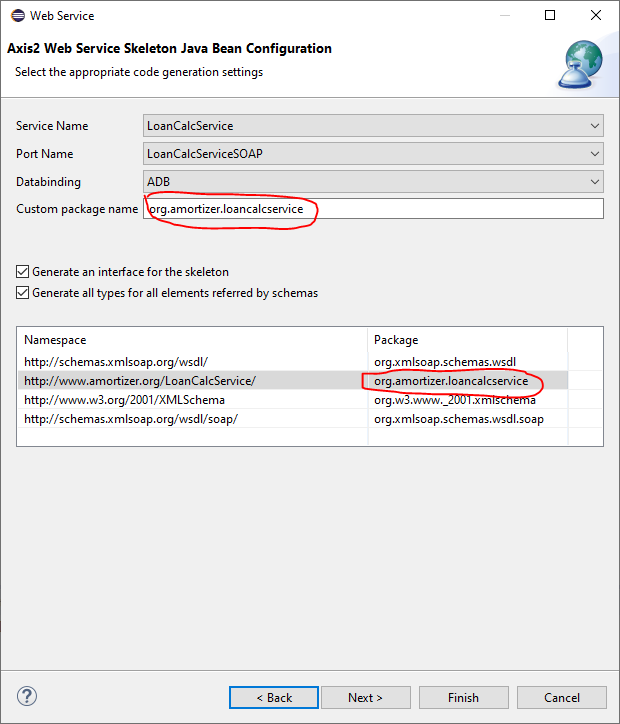
Check 'Generate an interface for the skeleton' and 'Generate all types for all elements...'

The Custom package name is skewed by Axis2 so it needs to be corrected.

Update Custom package name as: org.amortizer.loancalcservice

and

3rd row 2nd col as: org.amortizer.loancalcervice



Select 'Finish' to generate the service code.

You can either proceed with starting the Tomcat service to publish the newly created service or cancel out since the code has already been generated.

At this point you have a single package under the src folder named:

org.amortizer.loancalcservice

A second package was manually created named org.amortizer.loancalcservice.loancalcs that contains the LoanCalculator class.

To wire the service methods to the calculator class, the LoanCalcServiceSkeleton class is updated for each method defined by the WSDL interface. At present 4 of the 6 methods in the WSDL are mapped.

**Client Generation**

The client is generated in a similar fashion but when adding the Web Service element to the project select Web Service Client. Use the WSDL and modify the package name (pg 10).

Once the client code is generated a org.amortizer.locancalcservice package is created that contains 2 classes:

LoanCalcServiceCAllbackHandler

LoanCalcServiceStub

The stub class is what is used to make the calls from the client to the service.

Although I used a swing app to make the call, the same call could be made from a jsp page, or C++,C#, Python, Perl, and probably Javascript.