Exercise 4

Creating a JAX-WS Service

Prior Knowledge
Basic understanding of SOAP and WSDL

Objectives
Understand how to create Web Services in Java

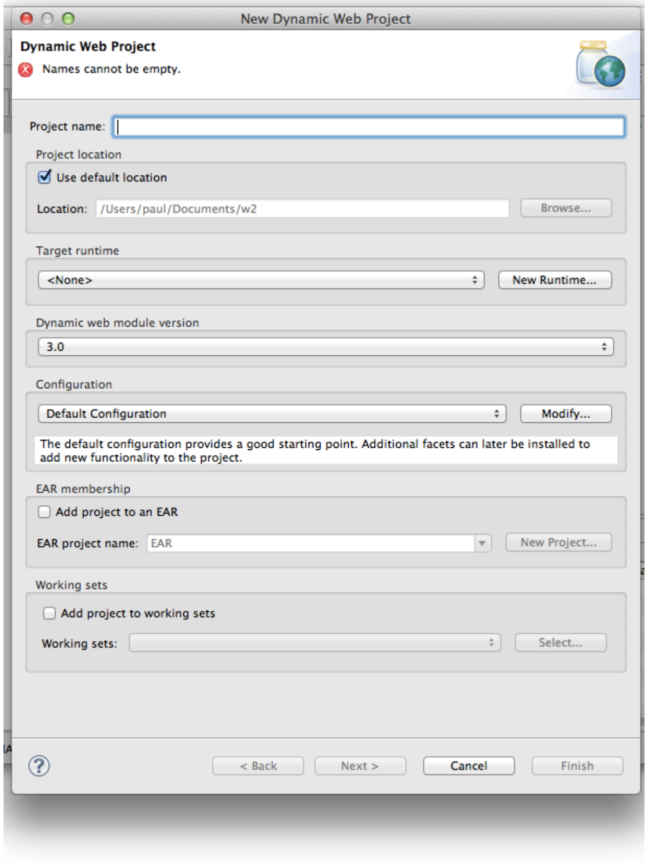
Software Requirements

- Java Development Kit 7
- Tomcat 7.0.69
- Eclipse JEE Luna workbench
- Apache CXF 2.7.18
- SOAPUI

Steps:

1. Make sure Tomcat is installed (e.g. in ~/servers/tomcat)
2. Check Apache CXF is installed (in ~/servers/apache-cxf-2.7.18/)
3. Open up Eclipse (from the Ubuntu Launcher)

4. Create a new Dynamic Web Project (**File -> New -> Dynamic Web Project**) (if this isn't visible, it may be you are on a different perspective – then choose Other and search for Dynamic).


New Dynamic Web Project

Dynamic Web Project

Names cannot be empty.

Project name:

Project location

[✓] Use default location

Location: /Users/paul/Documents/w2

Target runtime: <None>

Dynamic web module version: 3.0

Configuration: Default Configuration

The default configuration provides a good starting point. Additional facets can later be installed to add new functionality to the project.

EAR membership

[ ] Add project to an EAR

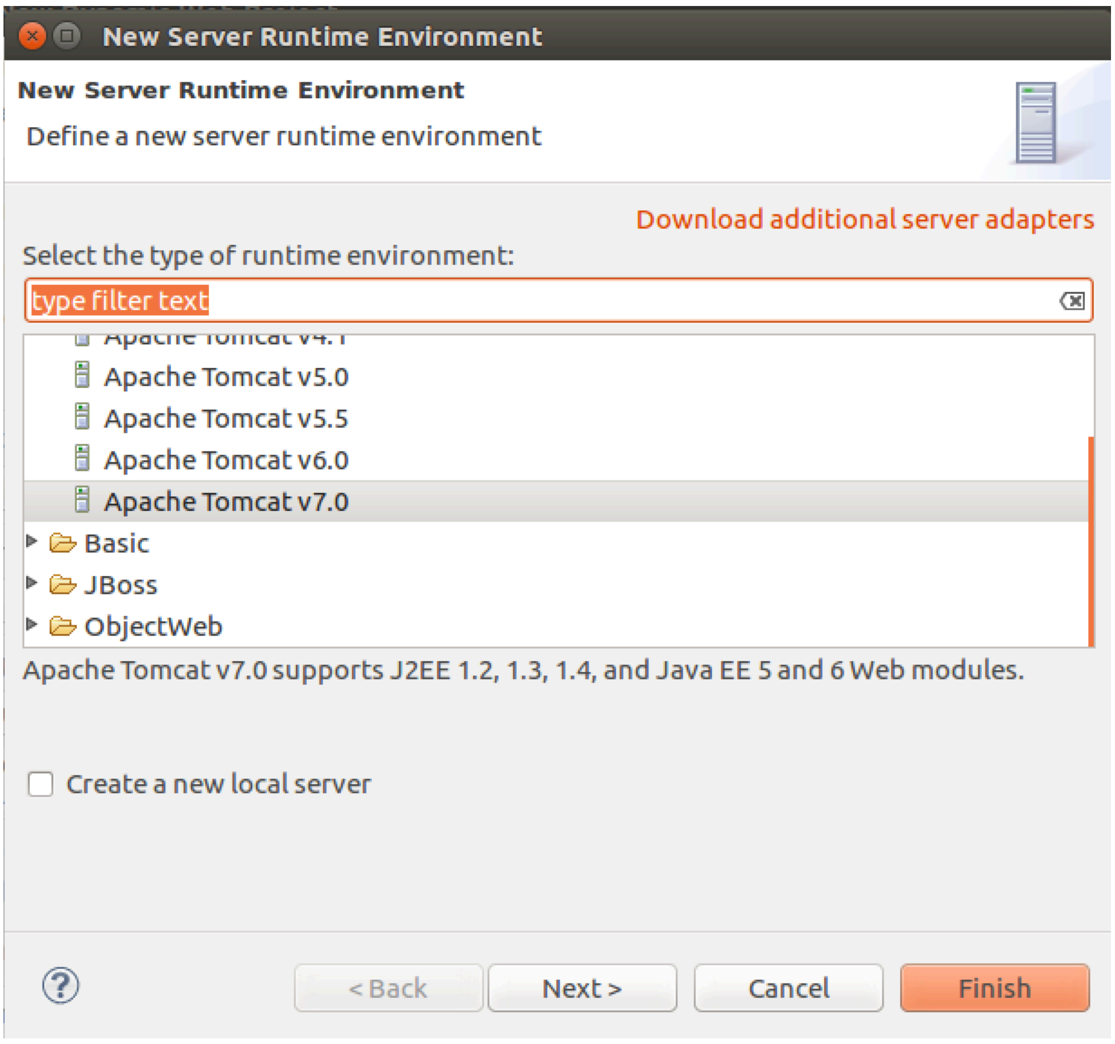
EAR project name: EAR

Working sets

[ ] Add project to working sets

Working sets:

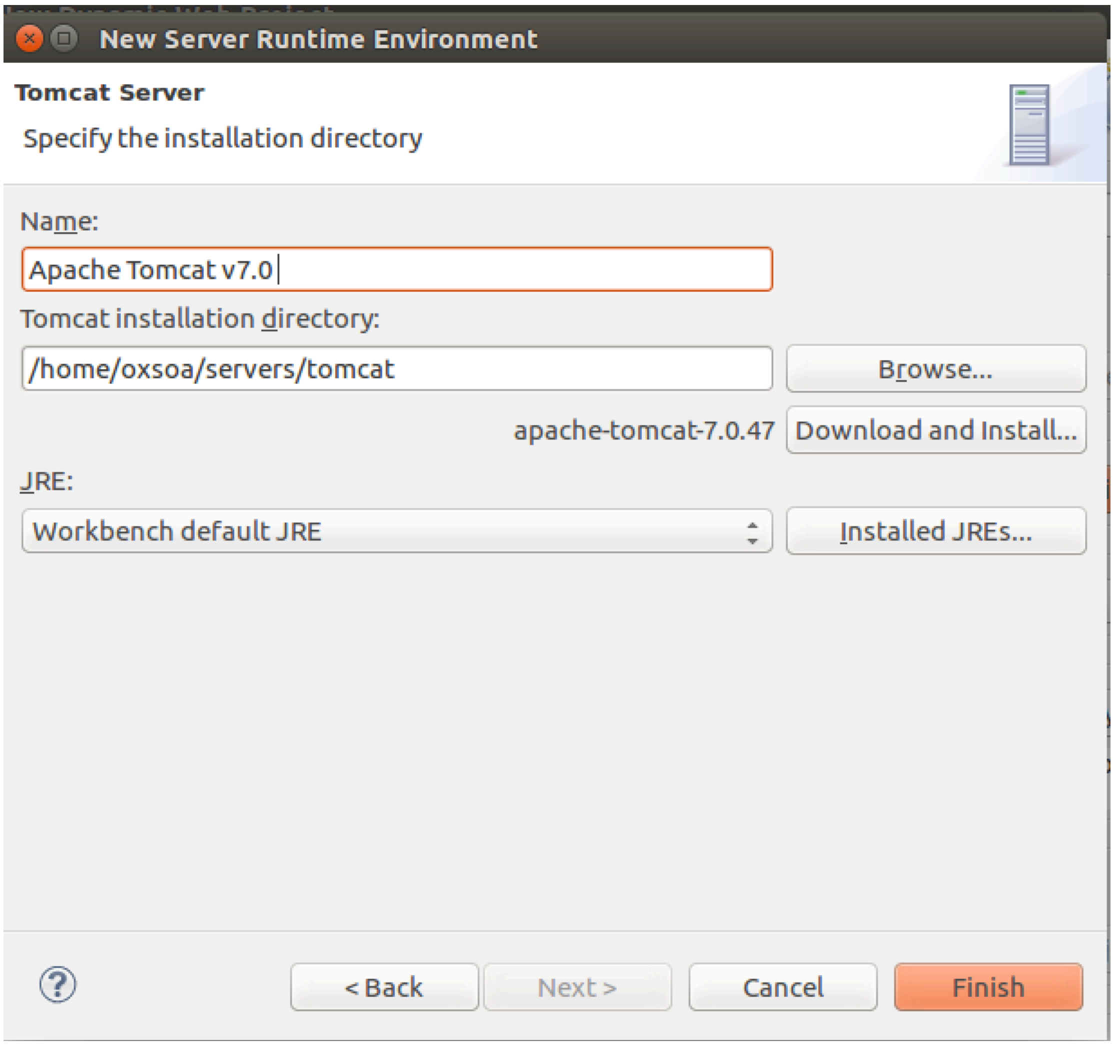
5. Now click **New Runtime** and choose Tomcat 7.0


New Server Runtime Environment
New Server Runtime Environment
Define a new server runtime environment
Download additional server adapters
Select the type of runtime environment:
type filter text
Apache Tomcat v5.0
Apache Tomcat v5.5
Apache Tomcat v6.0
Apache Tomcat v7.0
Basic
JBoss
ObjectWeb
Create a new local server
< Back
Next >
Cancel
Finish

Apache Tomcat v7.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5 and 6 Web modules.

6. Click **Next >**

7. Browse to the directory where you have Apache Tomcat installed and select that:



New Server Runtime Environment

Tomcat Server

Specify the installation directory

Name: Apache Tomcat v7.0

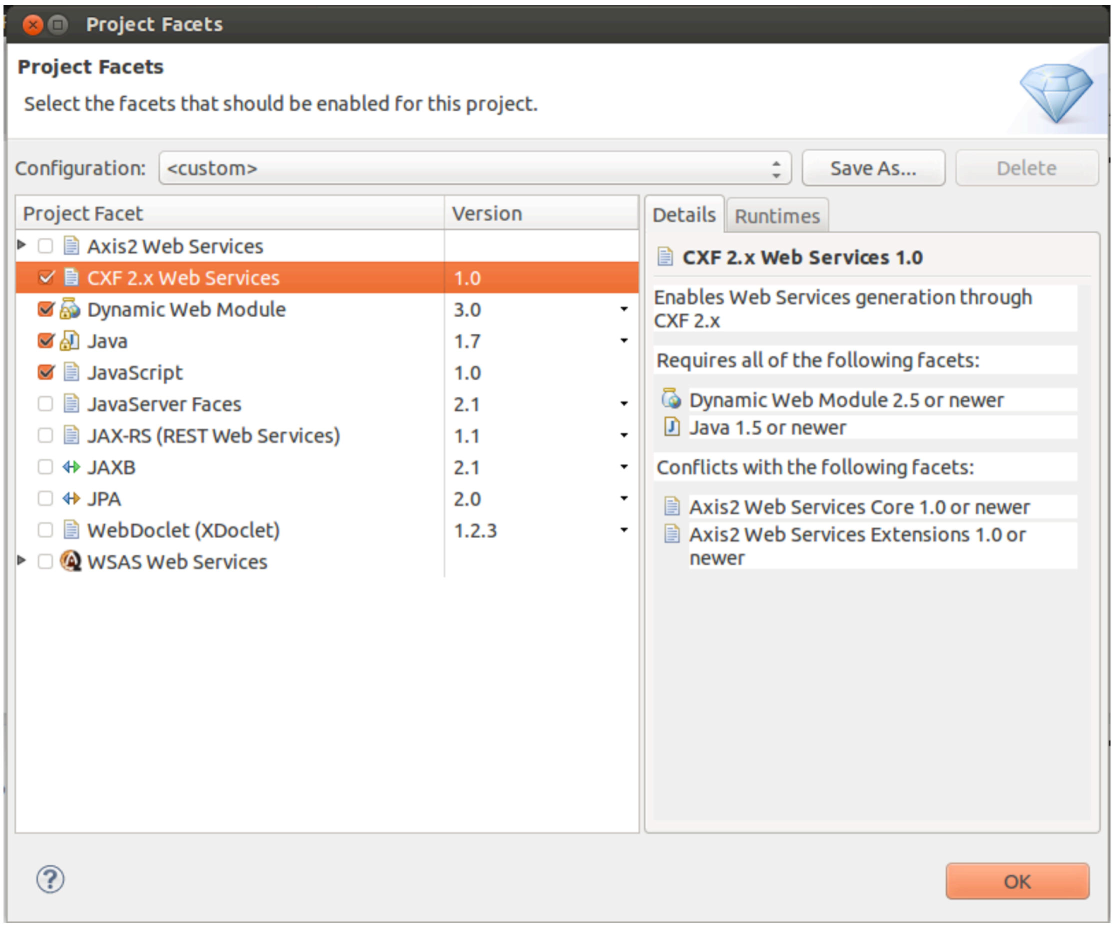
Tomcat installation directory: /home/oxsoa/servers/tomcat

apache-tomcat-7.0.47

JRE: Workbench default JRE

Click **Finish**

8. Now we need to modify the Tomcat configuration. Click on **Modify** next to Default Tomcat Configuration:



Project Facet	Version
☐ Axis2 Web Services	
☑ CXF 2.x Web Services	1.0
☑ Dynamic Web Module	3.0
☑ Java	1.7
☑ JavaScript	1.0
☐ JavaServer Faces	2.1
☐ JAX-RS (REST Web Services)	1.1
☐ JAXB	2.1
☐ JPA	2.0
☐ WebDoclet (XDoclet)	1.2.3
☐ WSAS Web Services	

CXF 2.x Web Services 1.0

Enables Web Services generation through CXF 2.x

Requires all of the following facets:

- Dynamic Web Module 2.5 or newer
- Java 1.5 or newer

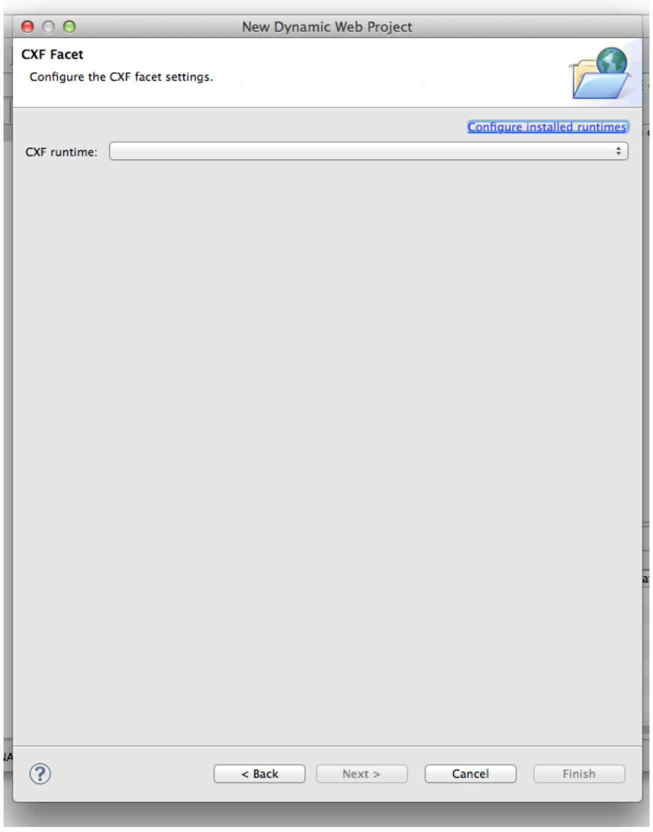
Conflicts with the following facets:

- Axis2 Web Services Core 1.0 or newer
- Axis2 Web Services Extensions 1.0 or newer

Enable the **CXF 2.x Web Services** section. Click **OK**

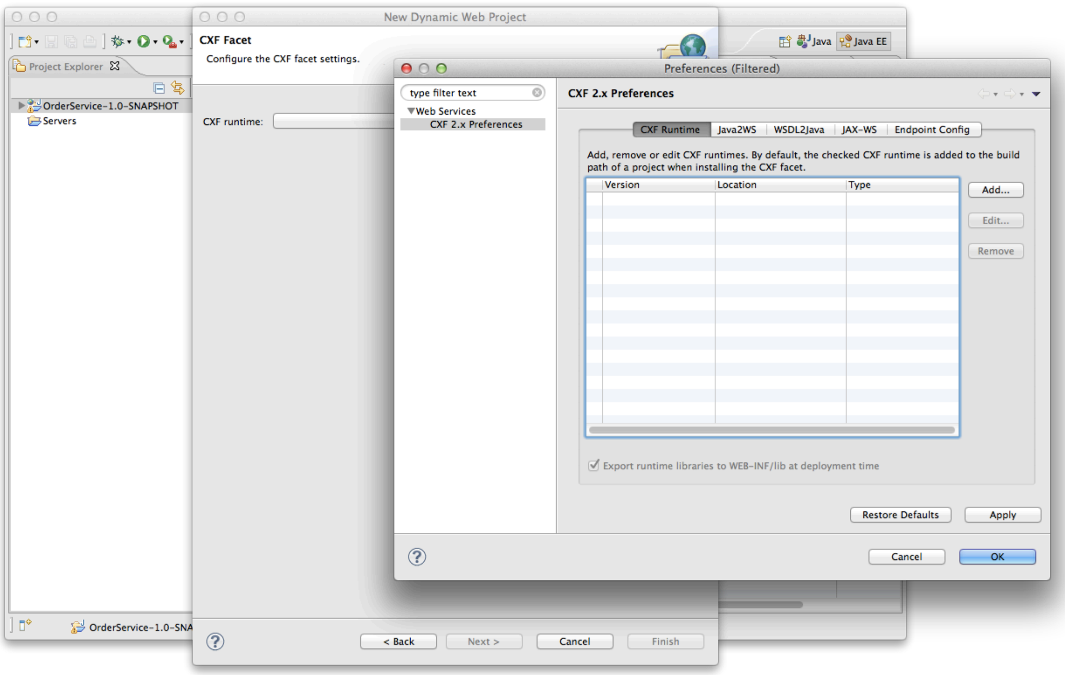
9. Now you should be back in the initial Dialog, so give the project a name, e.g. **JAXWSSample**

10. Click **Next, Next, Next** until you are at the CXF Facet configuration dialog



11. If the CXF 2.7.18 runtime is already configured you can skip the next two steps.

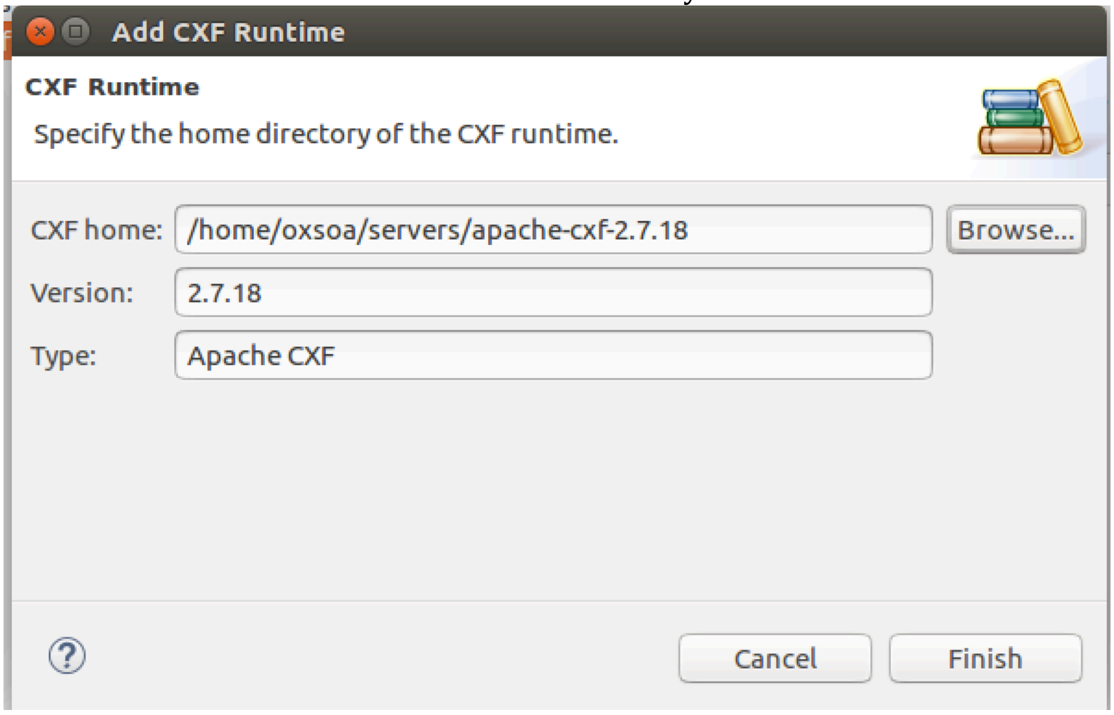
12. Click **Configure Installed Runtimes**



Add, remove or edit CXF runtimes. By default, the checked CXF runtime is added to the build path of a project when installing the CXF facet.

✓ Export runtime libraries to WEB-INF/lib at deployment time

13. Now click **Add.** Browse to the CXF install directory.



Add CXF Runtime

CXF Runtime

Specify the home directory of the CXF runtime.

CXF home: /home/oxsoa/servers/apache-cxf-2.7.18

Version: 2.7.18

Type: Apache CXF

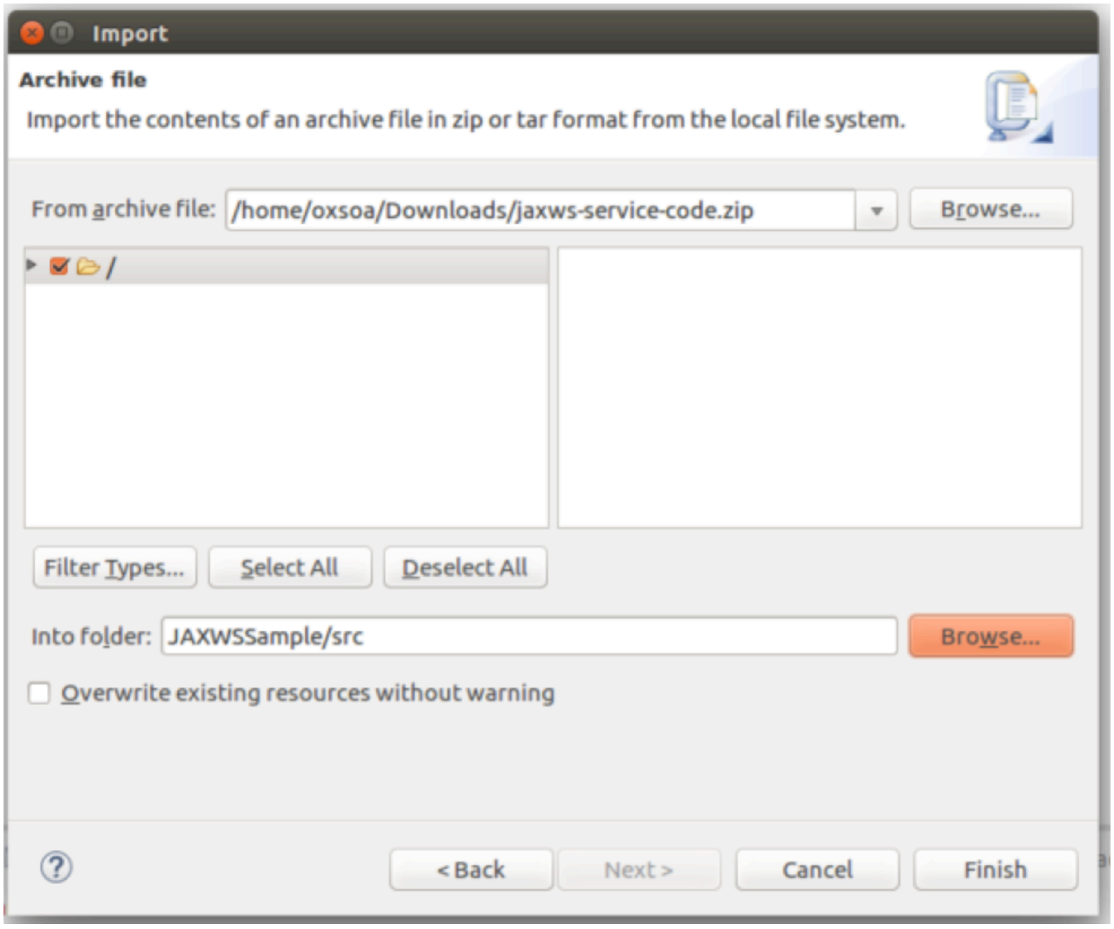
Now click **Finish**

14. Now **select the tick box** next to the CXF 2.7.18 version.

15. Click **OK** and **Finish**. You should now have a project.

16. Import some existing code I have written:
    File->Import / General -> Archive File
    Browse to /home/oxsoa/Downloads/jaxws-service-code.zip

    Change the *Into Folder* to be: **JAXWSSample/src**



Import

Archive file

Import the contents of an archive file in zip or tar format from the local file system.

From archive file: /home/oxsoa/Downloads/jaxws-service-code.zip

Into folder: JAXWSSample/src

☐ Overwrite existing resources without warning

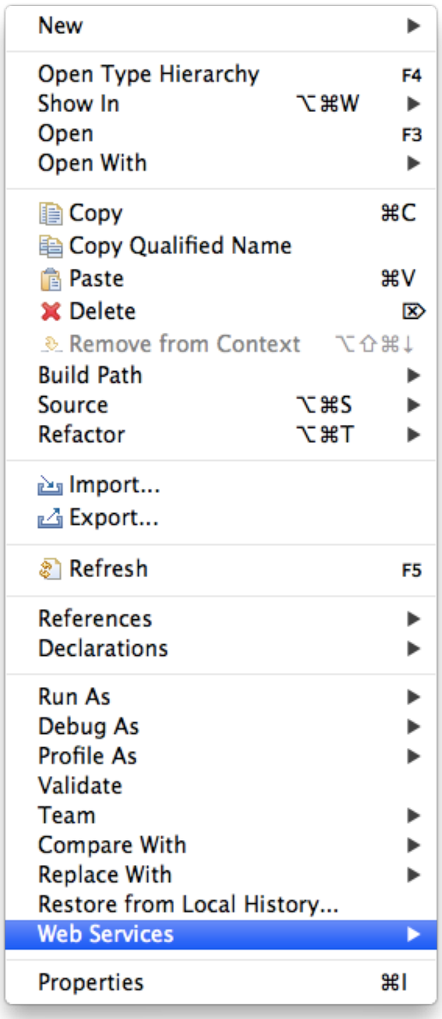
17. The package includes the following files:

    freo.me.jaxws.Order
    freo.me.jaxws.OrderService
    freo.me.jaxws.OrderServiceImpl
    freo.me.jaxws.NotFoundException

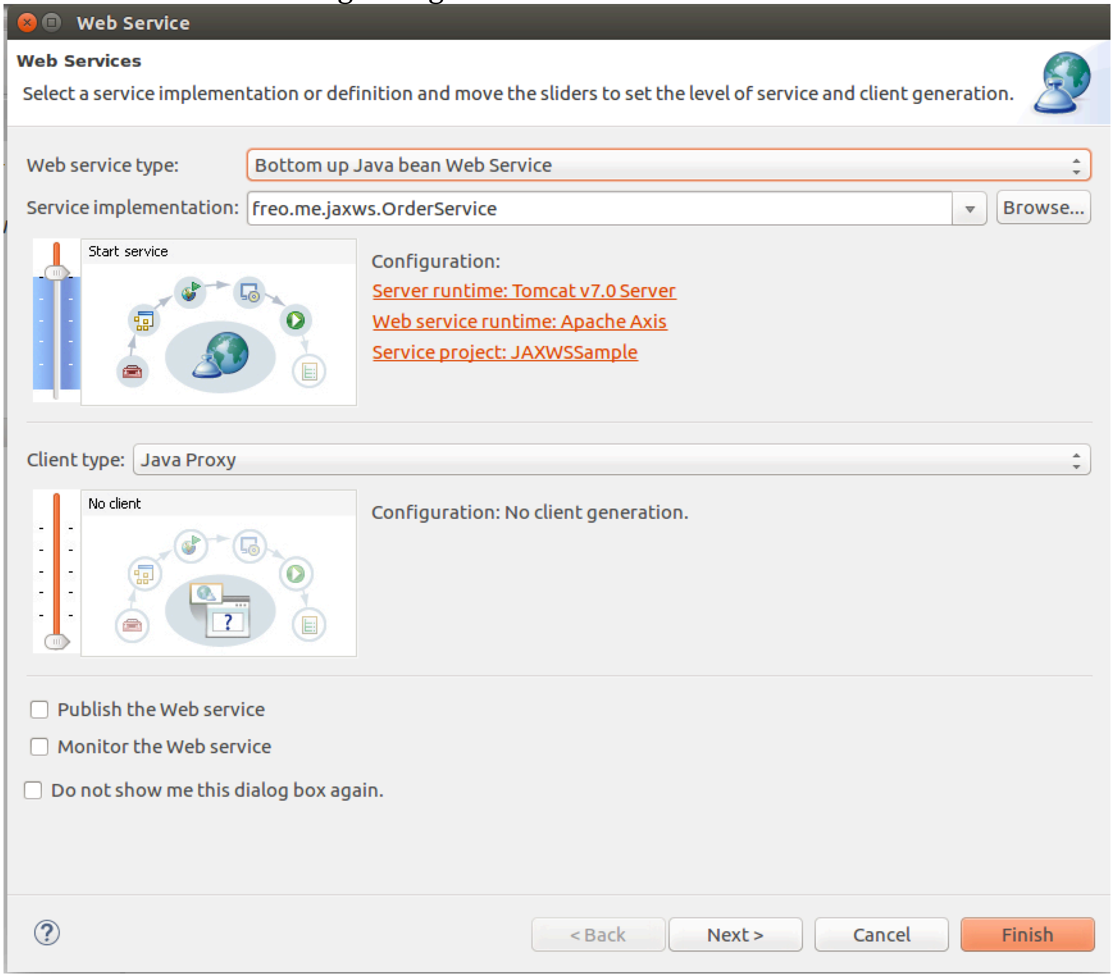
    The code is pure Java and therefore you should be able take a good look at the code and understand it.

    Now we can expose this code as a Web Service using the JAXWS wizard built into Eclipse.

18. Select the **OrderService** class and Right-Click on it. Now Select **Web Services**. Then **Create Web Service**.



19. You will see the following dialog



Select a service implementation or definition and move the sliders to set the level of service and client generation.

Web service type: Bottom up Java bean Web Service

Service implementation: freo.me.jaxws.OrderService

Configuration:
Server runtime: Tomcat v7.0 Server
Web service runtime: Apache Axis
Service project: JAXWSSample

Client type: Java Proxy

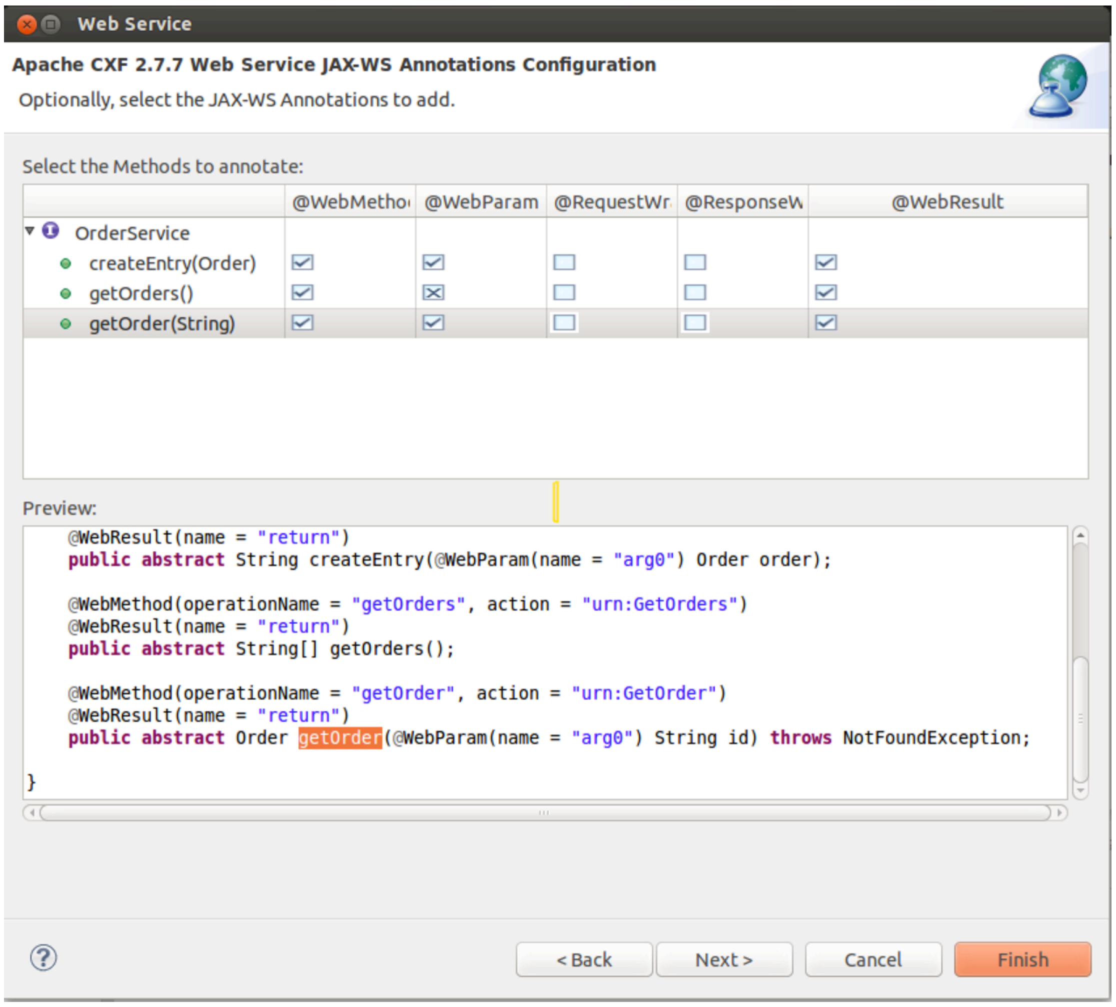
Configuration: No client generation.

☐ Publish the Web service
☐ Monitor the Web service
☐ Do not show me this dialog box again.

20. Now Click on the **Web Service Runtime** and change it to **Apache CXF v2.x.** Click **Next>**

21. Choose **freo.me.jaxws.OrderServiceImpl** as the Service Implementation. Click **Next>**

Click **Next>**
You will see a screen like this:



Apache CXF 2.7.7 Web Service JAX-WS Annotations Configuration

Optionally, select the JAX-WS Annotations to add.

Select the Methods to annotate:

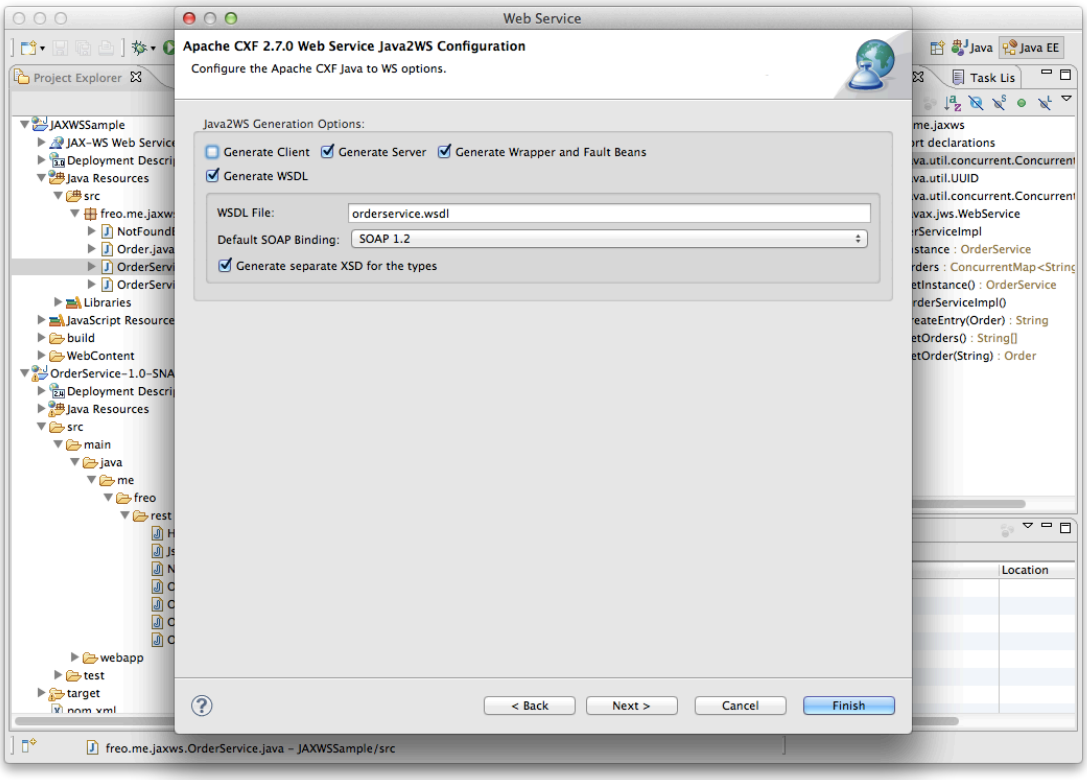
	@WebMetho	@WebParam	@RequestWr	@ResponseW	@WebResult
OrderService					
createEntry(Order)	✓	✓	☐	☐	✓
getOrders()	✓	☒	☐	☐	✓
getOrder(String)	✓	✓	☐	☐	✓

Preview:

```
    @WebResult(name = "return")
    public abstract String createEntry(@WebParam(name = "arg0") Order order);

    @WebMethod(operationName = "getOrders", action = "urn:GetOrders")
    @WebResult(name = "return")
    public abstract String[] getOrders();

    @WebMethod(operationName = "getOrder", action = "urn:GetOrder")
    @WebResult(name = "return")
    public abstract Order getOrder(@WebParam(name = "arg0") String id) throws NotFoundException;

}
```

For each method, select the **WebMethod**, **WebParam** and **WebResult** boxes and remove the **Request Wrapper** and **Response Wrapper** checkmarks as shown.
Then click **Next>**.

22. You should see:


Web Service

Apache CXF 2.7.0 Web Service Java2WS Configuration

Configure the Apache CXF Java to WS options.

Java2WS Generation Options:

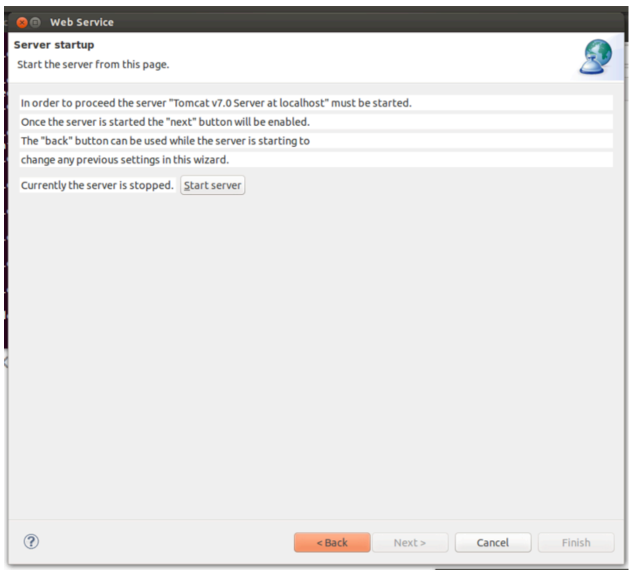
- [ ] Generate Client
- [✓] Generate Server
- [✓] Generate Wrapper and Fault Beans
- [✓] Generate WSDL

WSDL File: orderservice.wsdl

Default SOAP Binding: SOAP 1.2

- [✓] Generate separate XSD for the types

Now make sure Generate Server is clicked. Click **Next>**


Web Service

Server startup

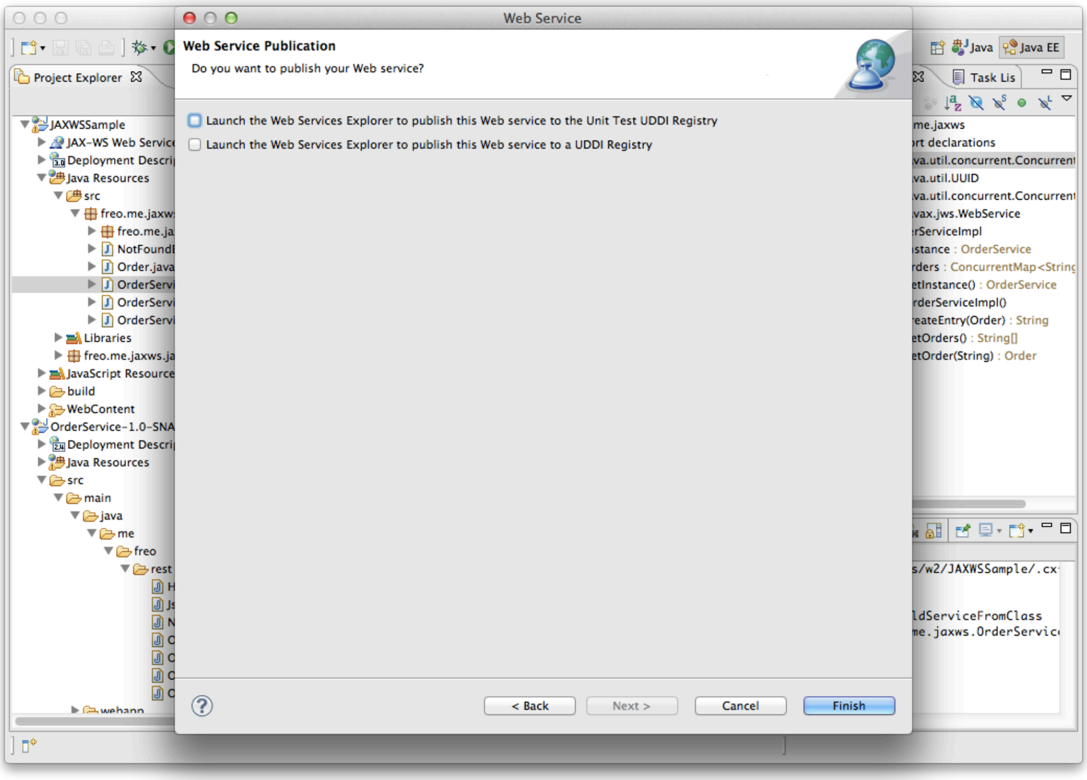
Start the server from this page.

In order to proceed the server "Tomcat v7.0 Server at localhost" must be started.
Once the server is started the "next" button will be enabled.
The "back" button can be used while the server is starting to
change any previous settings in this wizard.

Currently the server is stopped. Start server

Click **Start Server**
Hint: if the server fails to start, you may still be running another server (e.g. Tomcat or Node) on port 8080. Close those down.
Then **Next**

23. Do **not** select either UDDI option:

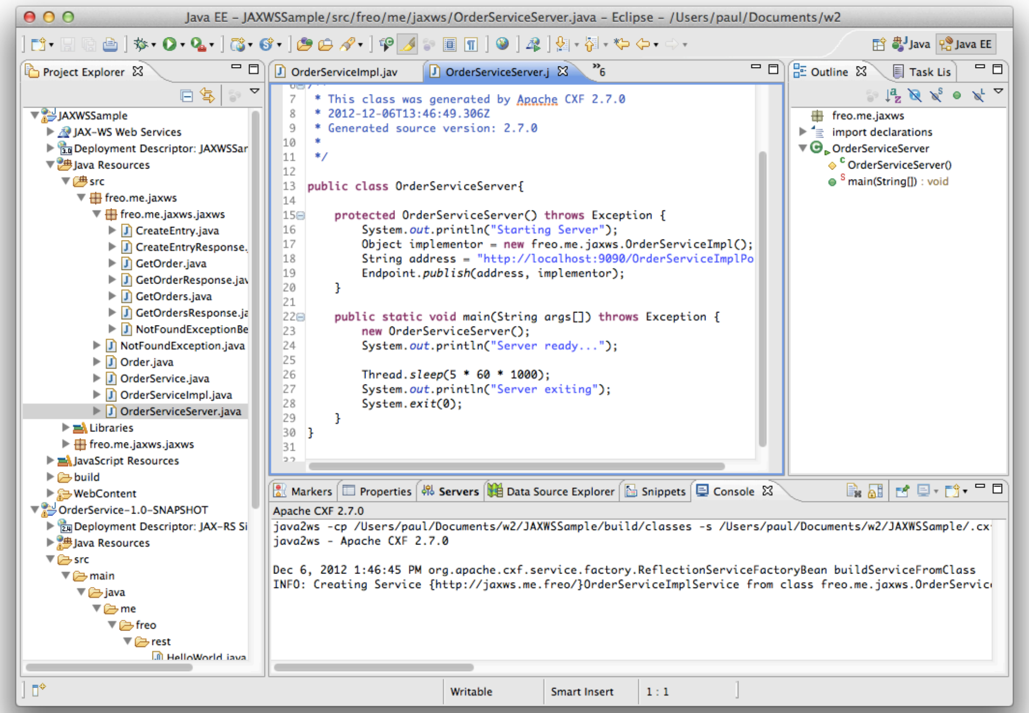


Launch the Web Services Explorer to publish this Web service to the Unit Test UDDI Registry

Launch the Web Services Explorer to publish this Web service to a UDDI Registry

24. Click **Finish**

25. This will generate a number of additional classes.



```
 * This class was generated by Apache CXF 2.7.0
 * 2012-12-06T13:46:49.306Z
 * Generated source version: 2.7.0
 *
 */

public class OrderServiceServer{

    protected OrderServiceServer() throws Exception {
        System.out.println("Starting Server");
        Object implementor = new freo.me.jaxws.OrderServiceImpl();
        String address = "http://localhost:9090/OrderServiceImplPo
        Endpoint.publish(address, implementor);
    }

    public static void main(String args[]) throws Exception {
        new OrderServiceServer();
        System.out.println("Server ready...");

        Thread.sleep(5 * 60 * 1000);
        System.out.println("Server exiting");
        System.exit(0);
    }
}
```

```
Apache CXF 2.7.0
java2ws -cp /Users/paul/Documents/w2/JAXWSSample/build/classes -s /Users/paul/Documents/w2/JAXWSSample/.cx
java2ws - Apache CXF 2.7.0

Dec 6, 2012 1:46:45 PM org.apache.cxf.service.factory.ReflectionServiceFactoryBean buildServiceFromClass
INFO: Creating Service {http://jaxws.me.freo/}OrderServiceImplService from class freo.me.jaxws.OrderServic
```

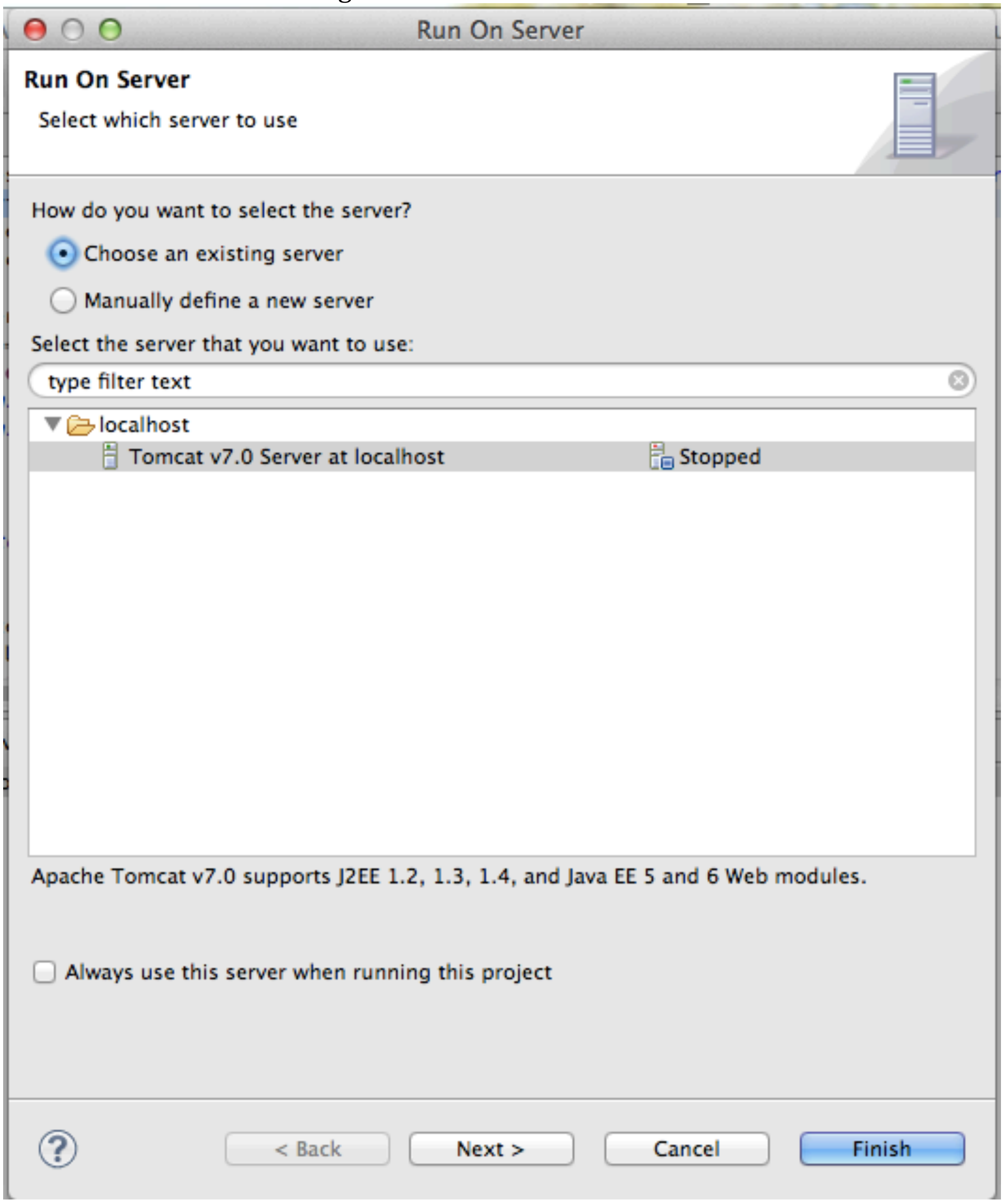
Firstly, it will create a set of classes to map XML types. Secondly it will create a set of annotations in your existing code. Finally it will create a Server class that can be used to run the service standalone.

26. Look through the code and identify all three parts. Take a look at the annotations.

27. You can now run this in one of three ways:
    * Using the generated server class
    * Deploying in Tomcat using Eclipse **[This should already be running!]**
    * Deploying in Tomcat standalone

28. We are going to focus on the second option.

29. To run Tomcat locally "within" Eclipse, choose the overall JAXWSSample app in the left hand Project Explorer, right-click and select **Run As->Run on Server**. You will see a dialog like this:



How do you want to select the server?

[✓ Choose an existing server]

Manually define a new server

Select the server that you want to use:

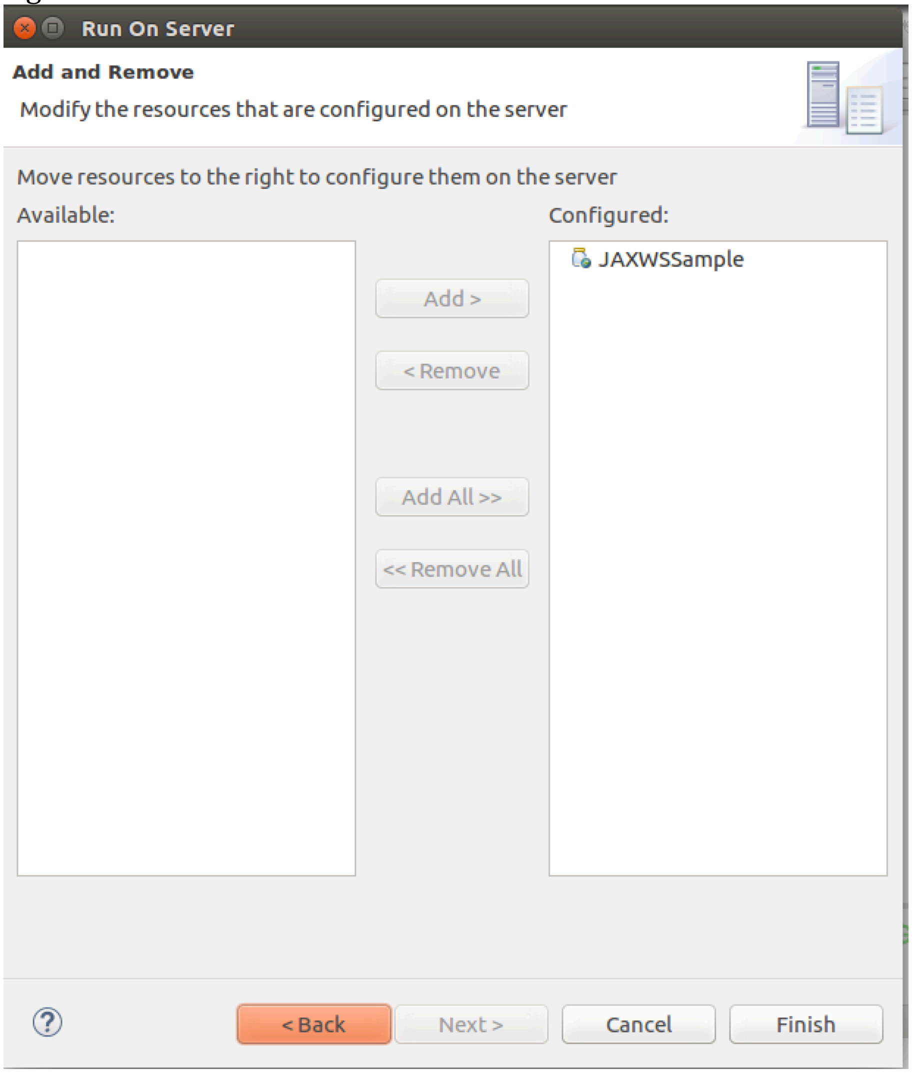
type filter text

localhost
Tomcat v7.0 Server at localhost   Stopped

Apache Tomcat v7.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5 and 6 Web modules.

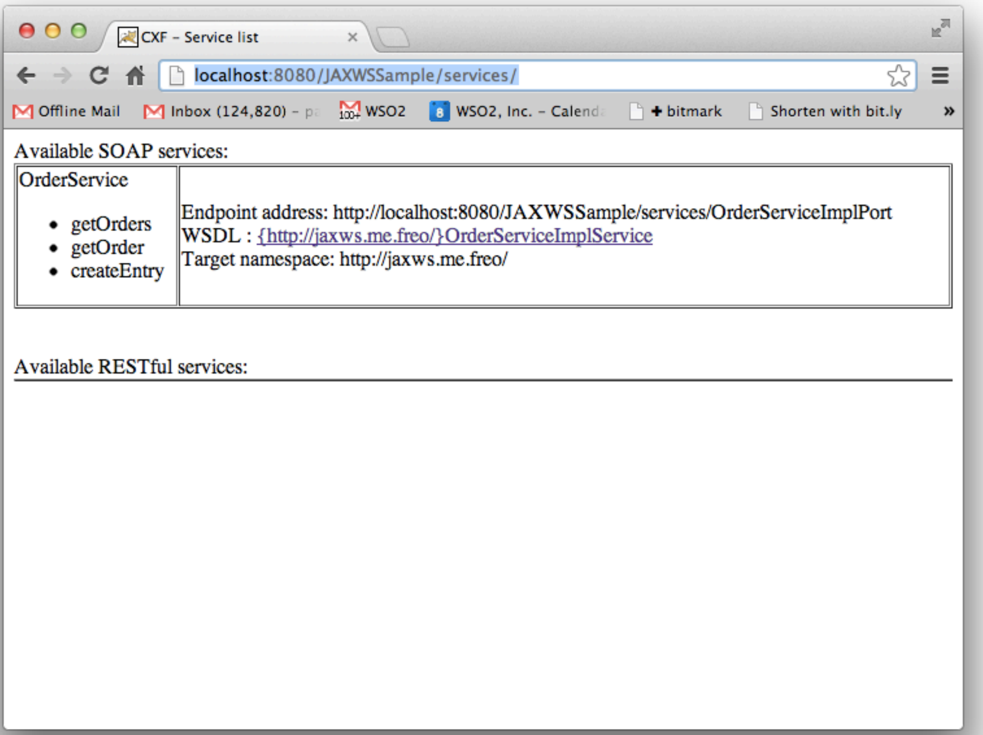
☐ Always use this server when running this project

30. Choose Tomcat 7.0 and click **Next>**
    You should see your JAXWSSample app in the Configured column on the right:



31. Click **Finish**
    The server will start up and Eclipse will try to start up a browser against your app. There will be a **404 Not Found** error because our WAR file has no "Welcome Page".

32. Browse http://localhost:8080/JAXWSSample/services/ and you should see a CXF generated page:



Available SOAP services:

OrderService • getOrders • getOrder • createEntry	Endpoint address: http://localhost:8080/JAXWSSample/services/OrderServiceImplPort WSDL : {http://jaxws.me.freo/}OrderServiceImplService Target namespace: http://jaxws.me.freo/

Available RESTful services:

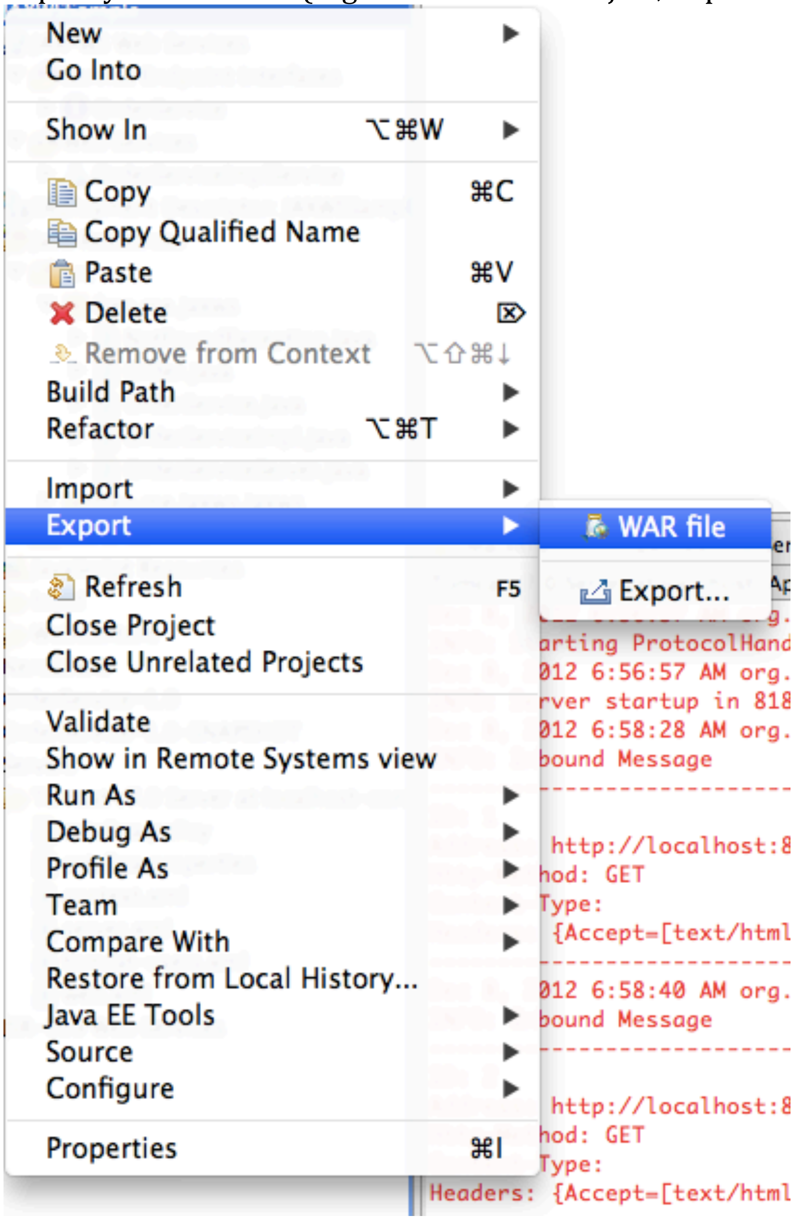
33. Click on the link and it should take you the URL.

34. Import this URL into SOAPUI and **test the service.**

35. That's it! (unless you fancy an extension).

Extension

Export your WAR file (Right Click on the Project, Export->WAR File)



```
arting ProtocolHand
012 6:56:57 AM org.
rver startup in 818
012 6:58:28 AM org.
bound Message
-------------------
 http://localhost:8
hod: GET
Type:
 {Accept=[text/html
-------------------
012 6:58:40 AM org.
bound Message
-------------------
 http://localhost:8
hod: GET
Type:
Headers: {Accept=[text/html
```

36. Stop the Server inside Eclipse, and copy your exported WAR file to Tomcat's **webapp** directory. Stop the server inside Eclipse

    Start Tomcat from the command line as before, and test again.

37. Congratulations