Setting up a WLS-Webservice Example in Eclipse

I like the API Examples that ship with WLS because they contain a level of complexity that is just enough to demonstrate the technology slice at hand. In other words, they are kept very simple. Therefore they are an excellent starting point for small test applications that can be used in very complex environments. We want to look at the Example: "Creating a Web Service from an Existing WSDL File". We demonstrate how to quickly integrate it into Eclipse with full support of code completion, context sensitive help and on-the-fly compilation support.

1 Contents

2 The Example from WLS

Let's have a quick look at the example itself. It is just a simple web service called "*TemperatureService*" that is called from a stand-alone java client, as depicted in the following diagram.

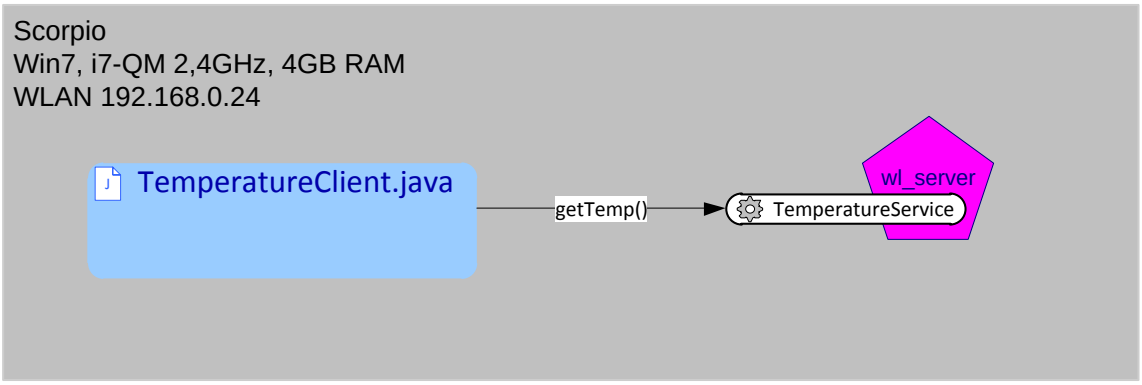

Figure 1. Web service Example with a stand-alone Java Client calling a web service on the example server.

This example demonstrates the build process of a web service starting from a wsdl file, which is depicted in the following diagram.

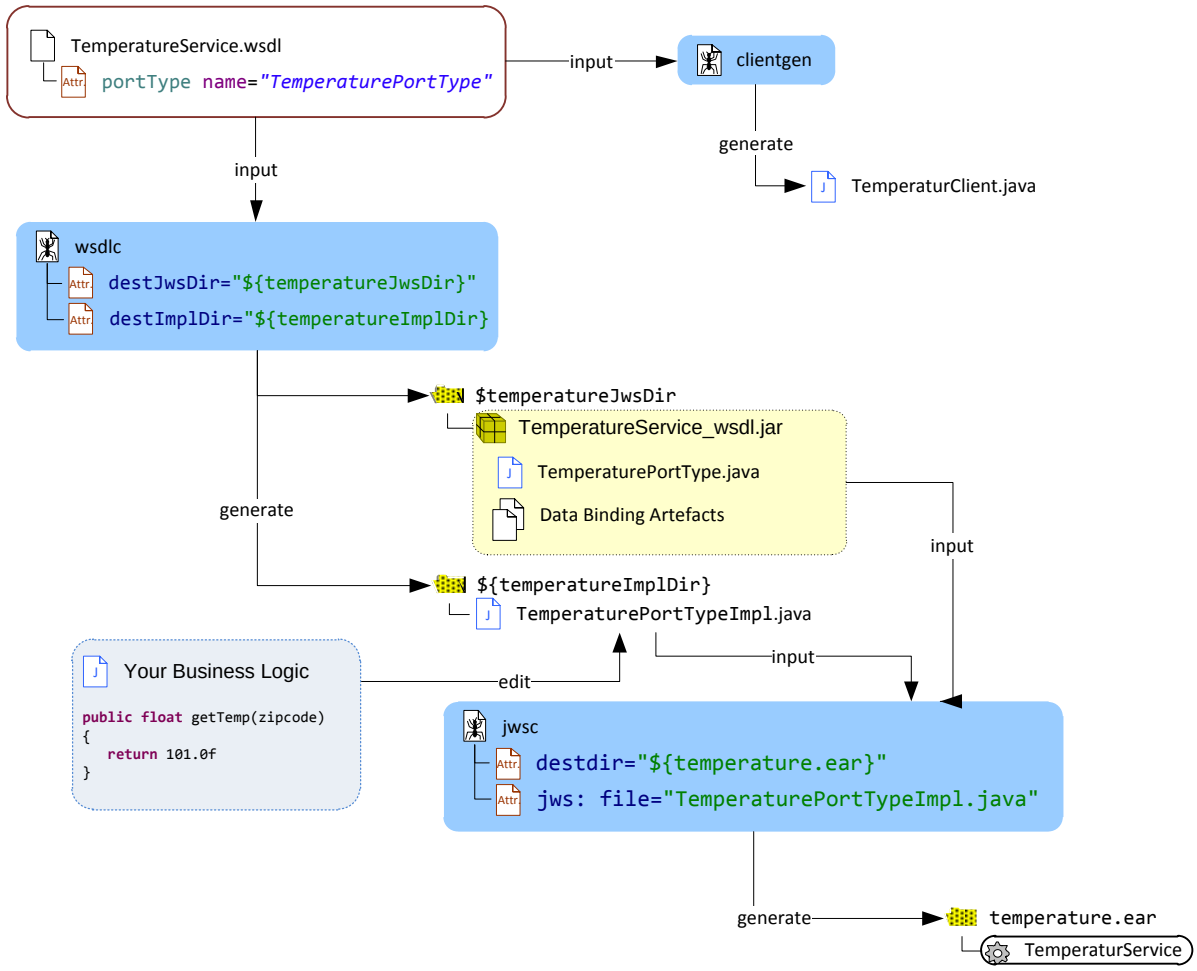

Figure 2. Build-Process of the web service.

The process starts from the file *TemperaturService.wsdl*. The ant task *wsdlc* generates a JWS file *TemperaturPortTypeImpl.java* which is a stubbed out web service template. We insert our business logic in this file. The *wsdlc* task also generates a JAR file containing additional artifacts, needed to compile the web service. These two elements comprise the input for the *jwsc* ant task which eventually generates the deployable web service. The file *TemperaturService.wsdl* also serves as input for the client which is generated and built by the *clientgen* ant target.

We find the documentation for this example on the help page of our example server http://localhost:7001/docs/server/examples/src/examples/webservices/wsdl2service/instructions.html?skipReload=true. For convenience we provide a link to a PDF copy here.

3 Setting up the Example in Eclipse

Now let's turn to the hands-on work. We want to set up the example in Eclipse to be able to run and modify it. However we do not change the build workflow because we want to demonstrate a quick solution. As we will see later there is a conceptual gap between the standard way of the Eclipse IDE to build web services, and the example from WLS. We can imagine the WLS example as an external build like it is normally used in software development projects, supported by cruise control or maven.

3.1 Creating a Java Project

In Eclipse we create a new Java Project with the new project wizard as depicted in the following figures.

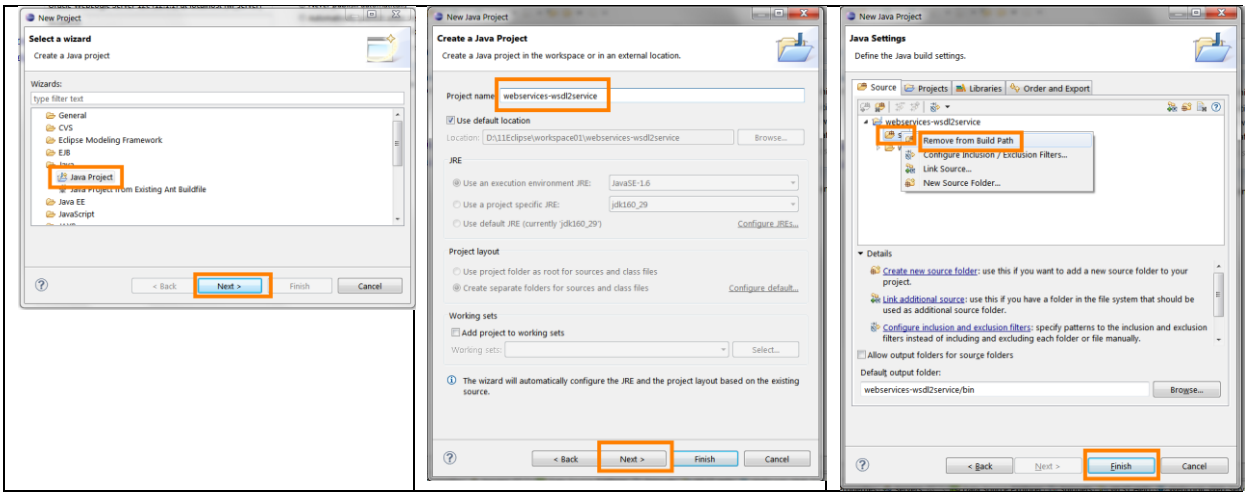

Figure 3. Dialog to create a new Project.

In the first step we choose the Java Project nature, since we want to use eclipse' JDT features. In the next step we provide the name *webservices-wsdl2service* for the project. We could choose any name but we stick to the proposal of the example documentation here.In the last step we remove the folder src from the build path because we want to set up our own built path from the pre-existing file structure of the example.

3.2 Importing the WLS example

In this step we will import only a single example from the collection of api examples. We therefore have to made some ajustments to the path settings.

We right-click on the new project in the package explorer and choose import to import the example files from weblogic samples directory. This is a two-step dialog as depicted below:

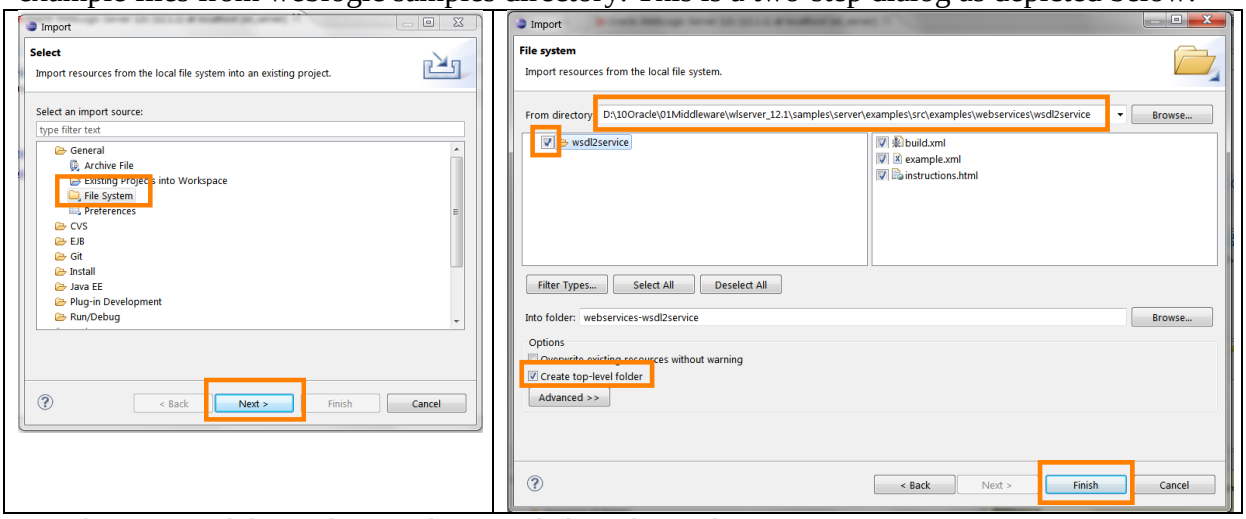

Figure 4. Dialog to import the sample into the projct.

In the first step we choose to import from the file system. In the next step we navigate to the example files of the WLS distribution. We select the directory for import and confirm that we want to use top-level folder and conclude the dialog.

In the same manner we import the file example.properties which currently not par t of the project.

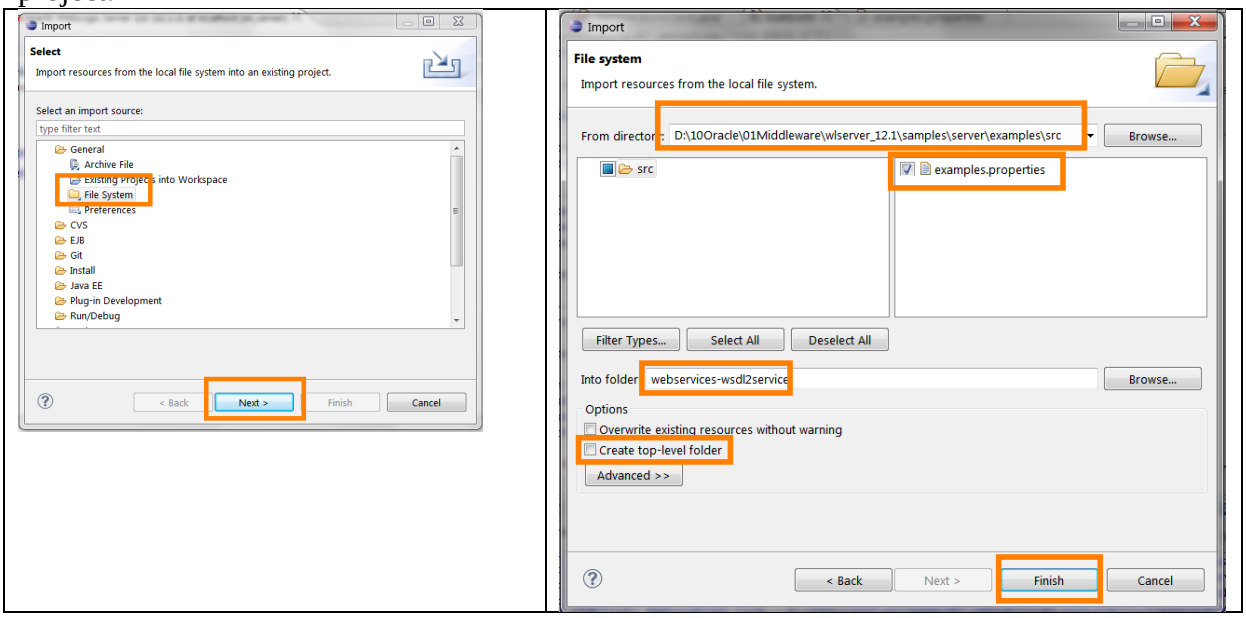

Figure 5. Importing example.properties into the project folder.

We navigate to the file and select it. This time we unselect “Create top-level folder” because we want the file to be placed directly in the project folder.
Now we have to make some adjustments to the example.properties and the build.xml files to reflect our slightly diverging file structure.
In the example.properties file we comment the property examples.home.dir and instead change it to a relative path, pointing to the parent folder, which is the basis for all ant operations.

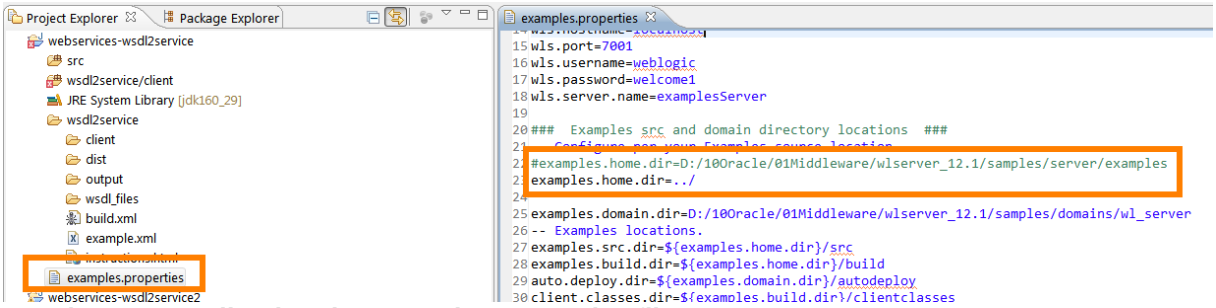

Figure 6. Adjusting the examples.properties file.

In the build.xml file we adjust the path which points to the example.properties file.

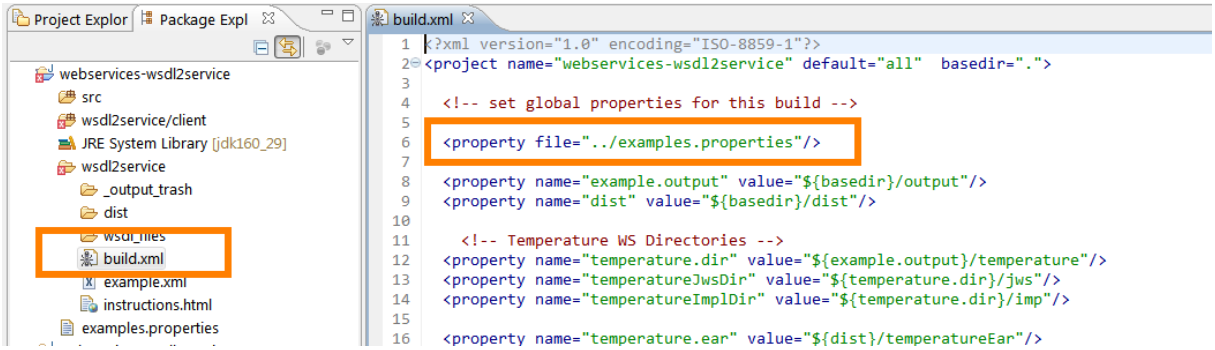

Figure 7. Adjusting the build.xml file.

As another prerequisite we need to import some ant tasks and create the server control pane for wl_server, but this is already set up from the last project. Please refer to.
http://weblogic-corner.blogspot.de/2012/03/wls12c-and-oracle-enterprise-pack.html
For convenience we display the ant runtime configuration with the additional tasks and their corresponding class definitions in the weblogi.jar file. It is available from the main menu "Window->Preferences".

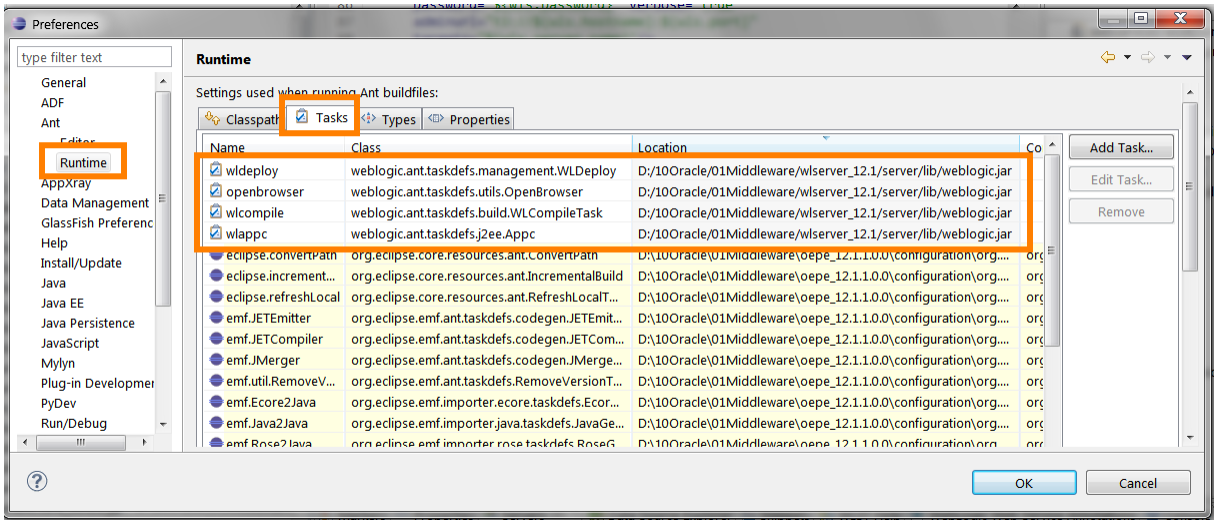

Figure 8. Adding additional ant tasks to eclipse.

Now we are ready to build, deploy and run the example.

3.3 Build and Run the example

Although there is support for web services in eclipse through the Web Tools Platform (http://www.eclipse.org/webtools/ws/) or through the OEPE (http://docs.oracle.com/cd/E14545_01/help/oracle.eclipse.tools.common.doc/html/index.html#webservices), we use neither of these options. We rather stick to the ant file is part of the imported example, since our goal is to demonstrate the integration of this example into eclipse.
Therefore we open the build.xml file in the project webservices-wsdl2service. If we switch to the Java EE Perspective, we should see the outline view on the right hand side. Otherwise we open the outline view from the "Window" menu. This is depicted in the figure below.

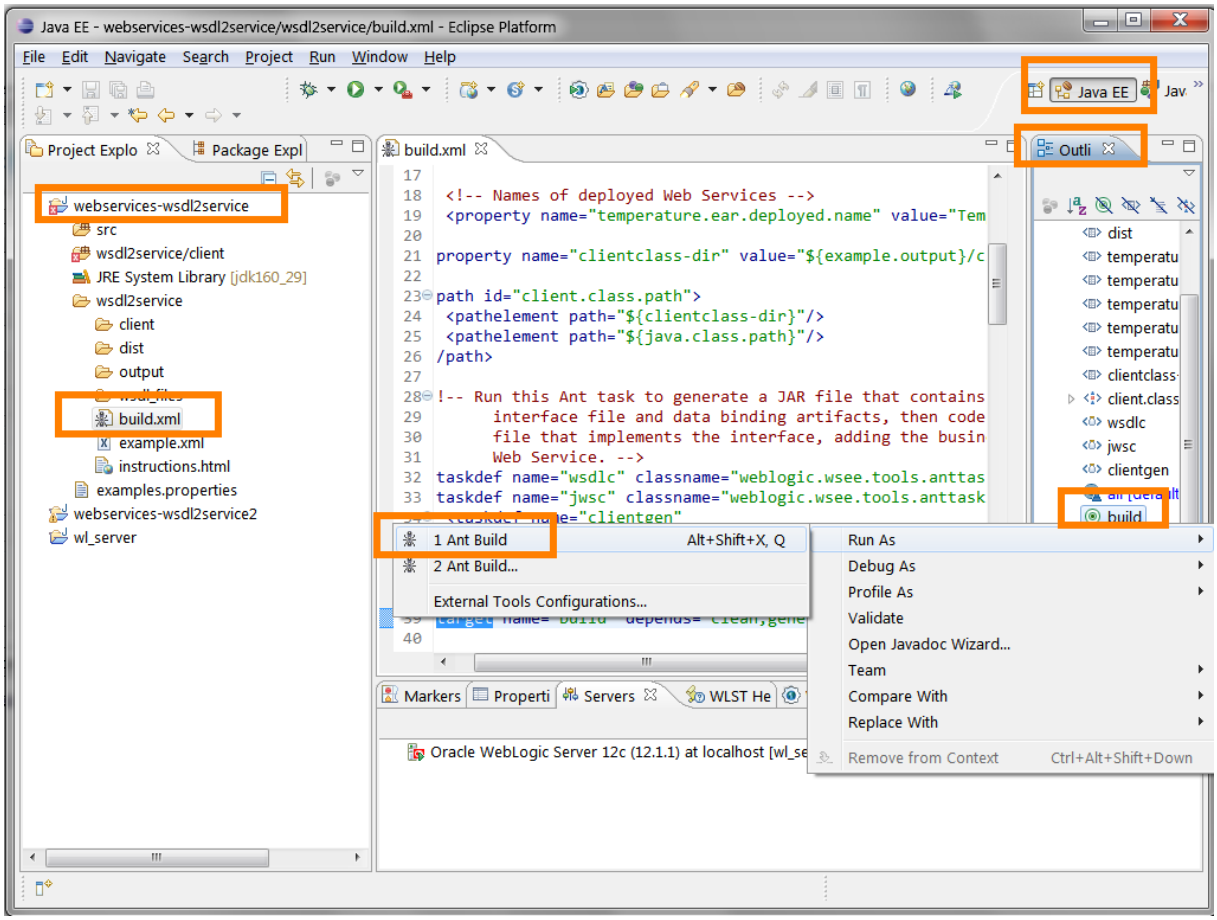

Figure 9. Using the ant script to build and run the project.

Now we right-click the target, we want to run and choose "Run As-> 1 Ant Build". This will build the project. We start the weblogic server in the server pane and then execute the ant target deploy in the same manner. Finally we execute the ant target "run" which will start the client and print output to the console pane.

If we now change the business logic by changing the return value to 101.0f in the implementation file of the web service as indicated by the picture below, we can use the ant file to go through the full edit-build-test cycle.

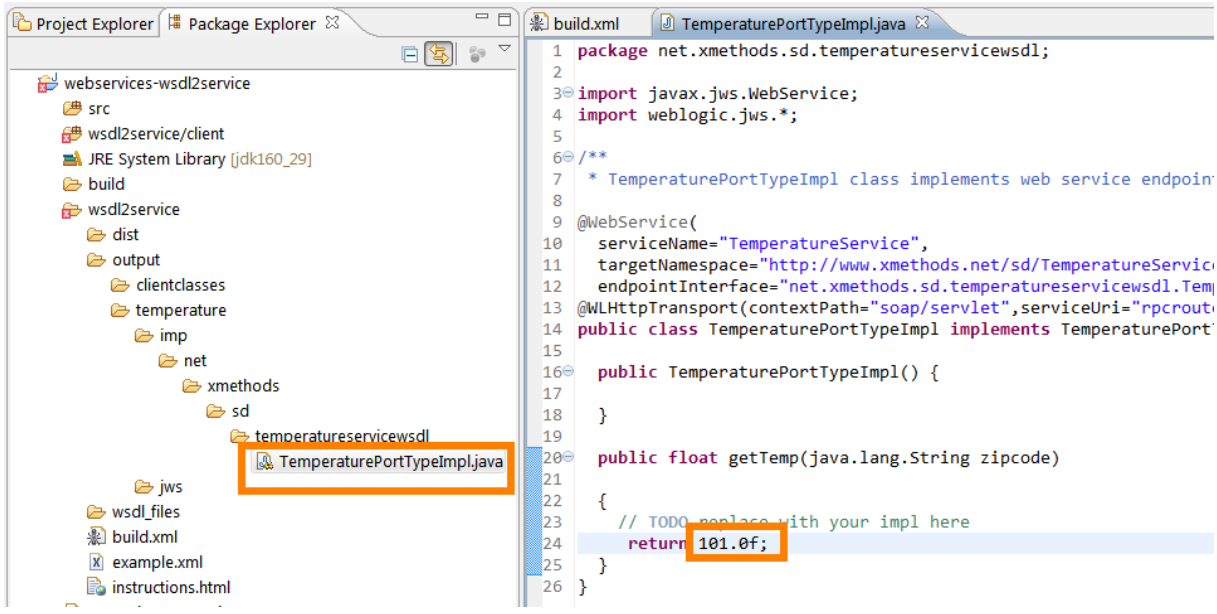

Figure 10. Editing the business logic of the web service.

After executing the ant targets build-redeploy-run, we observe that the client output reflects our change and prints out 101.1

3.4 Configuring on-the-fly compilation support

Although we do not use the JDT for building our project, we want to use the compiler support during editing. Therefore we have to adjust the Java Build Path. In a first step we include the library weblogic.jar, thus eliminating all include errors.

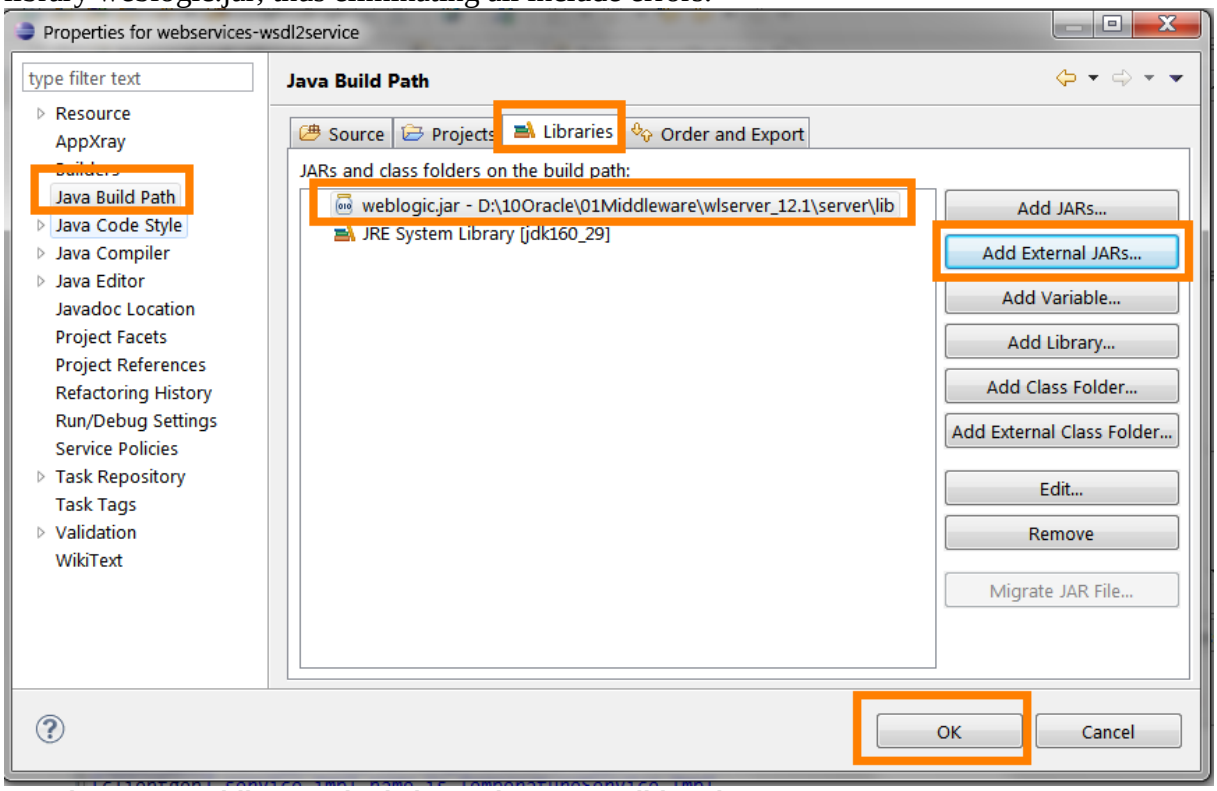

Figure 11. Adding weglogic.jar to the Java Build Path.

As indicated in the figure above, we open the project's property window and add the external library weblogic.jar to the Libraries tab of the Java Build Path.
Since we want to edit the file TemperaturPortTypeImpl.java we add the start folder of the package name to the "Source Folders" of the Java Build Path as indicated in the picture below.

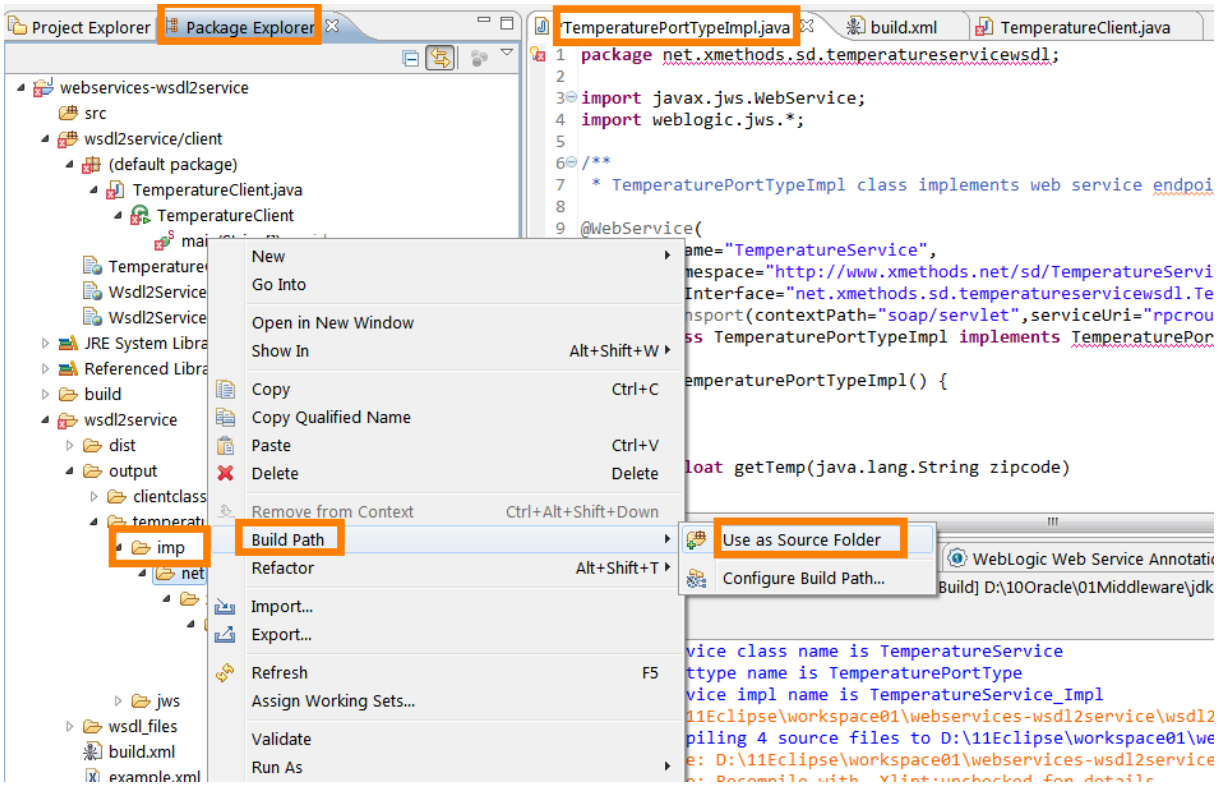

Figure 12. Adding a package folder to the source folder of eclipse.

Therefore we open the Package Explorer and expand the top folder of the folder tree we want to include. From the context menu we choose “Build Path->Use as Source Folder”
If we open the file TemperaturePortTypeImpl.java we now have compilation support and we recognize that the JDT complains about the compile error “TemperaturePortType cannot be resolved to a type”. This interface is generated with *wsdlc* and is included in the TemperaturService_wsdl.jar as show in the picture below. However we cannot add it to the build path libraries because the jar file only contains the uncompiled java class which is not recognized as java type.

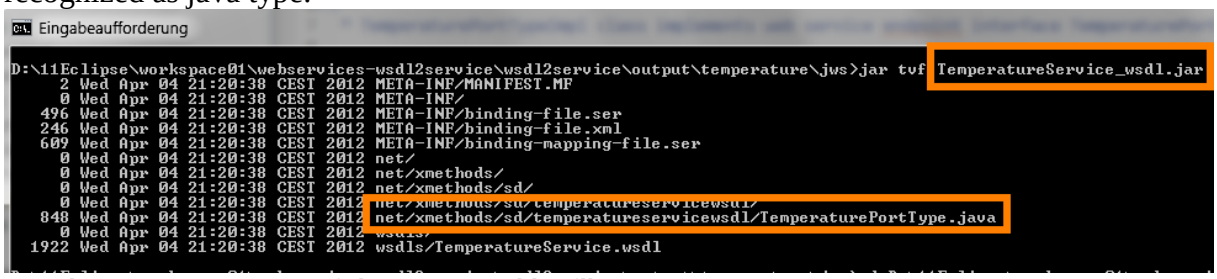

Figure 13. Contents of the generated jar file.

Alternatively we could unpack the jar file and add it to the source path. But we rather ignore the compile error.

3.5 Create an ear archive

If we want to deploy the web service on a remote weblogic server, it is more convenient to package it into an ear archive. Therefore we add an additional ant target to our build.xml file. We need to include the ant task wlpackge first which is depicted in the next figure.

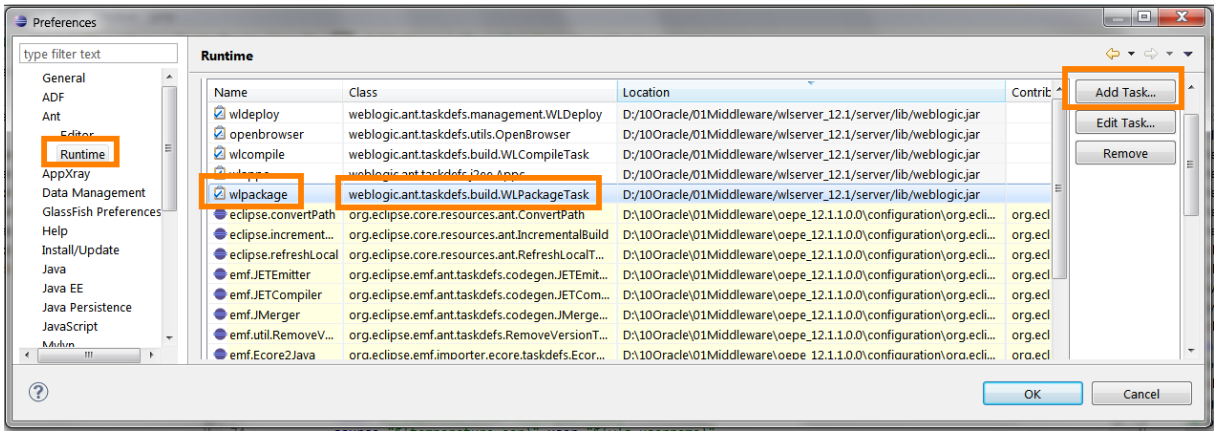

Figure 14. Including wlpackage to the ant runtime.

In the build.xml we add the following target, which can be executed after the build.service.ear and create deployable archive in the distribution directory.

```
<target name="build.service.ear">
        <wlpackage tofile="${dist}/temperatur.ear" srcdir="${temperature.ear}"
        destdir="${temperature.ear}" />
</target>
```

We can also add an additional target to deploy our ear file instead of the exploded version.

```
<target name="deploy.ear">
        <wldeploy action="deploy"
                name="${temperature.ear.deployed.name}"
                source="${dist}/temperatur.ear" user="${wls.username}"
                password="${wls.password}" verbose="true"
                adminurl="t3://${wls.hostname}:${wls.port}"
                targets="${wls.server.name}" />
</target>
```

We have to run this target manually because it is not included in the dependency list of the default target. We test it with the same client and it certainly produces the same result.

4 Conclusion

In this little workshop we showed how to integrate a simple web service example into an eclipse project which features context help, code assistant and on-the-fly compilation. Starting from a running example, which is part of the WLS installation, we produced an eclipse project which builds, deploys and runs this example. However instead of changing code with a text editor as it would be required in the original example, we now have the full IDE support. This enables us to quickly extend it to include code for prototypes and tests scenarios.

This workshop exemplifies an approach for taking any of the WLS examples and convert them into mini projects that could be deployed in complex environments for prototyping or analysis purposes.