Inhaltsverzeichnis

Inhaltsverzeichnis

WebServices with Java, Axis, XDoclet and Eclipse

Version 1.00

26. Februar 2004

Chapter 1. Introduction

Developing web applications in a J2EE enviroment and in particular web services is supported by very efficient tools, which do not even cost royalties.

With Eclipse as IDE, Tomcat or JBOSS as J2EE−Application−Server, Axis as WebServices Framework and XDoclet one has a powerful toolbox, which enable an efficient development process of web services.

This article will explain the setup of a complete environment and the needed plugins for Eclipse. Thus you can work fast and smoothly and concentrate on the actual business logic. All plugins will be introduced and a sample project will be developed, which implements a web services with Axis and Tomcat. A XDoclet template will be introduced for generating a deployment descriptor and an ant script for deployment. Finally you will see how to debug a web service in Eclipse.

Tools used: Eclipse 2.1.1, Tomcat 4.1.29, Sysdeo Tomcat Plugin 2.1.0, Lomboz−Plugin 2.1.2, Axis 1.1 and XDoclet 1.2 b3.

For WebService Security: VeriSign Trust Gateway 1.1 from 09/26/2003, ws−security 1.7. From IBM (for the additional security provider: IBM Emerging Technologies Toolkit Web Services 1.2. You do not have to download the tools from VeriSign and IBM because

the used Libraries are already in the sample project.

An advanced introduction to Java, J2EE, Tomcat or JBOSS is not the subject of this article. See the chapter Further Reading and additional sources in the Appendix.

Chapter 2. Eclipse: A modulare development environment

With Eclipse, IBM delivered a real masterpiece. Due to its modular design, Eclipse can be extended at will and thus be adapted to many different needs. As an open source product, it has a large popularity, especially due to the particularly large number of extensions. One can find a plugin [1], which makes life easier for all programming languages and typical developer`s task.

Since Eclipse is written in Java, it comes with full strengths especially in the Java environment where most plugins are also available.

2.1. Installation

Eclipse can be unpacked simply from archives and be installed in any path (with Windows e.g. `C:\Programme\Eclipse`) with Linux (Suse 8,1) in `/opt/eclipse` . Important for Suse 8.1: The library gtk 2,0 must be installed.

All plugins are in the `plugins` directory in the installation directory and herein the individual packages with reverse domain names as directories (e.g. `org.eclipse.jdt&`).

In order to install further plugins, you only need to unpack them in the directory hierarchy and restart Eclipse. New plugins will automatically be recognized and installation will be completed by Eclipse if nessecary.

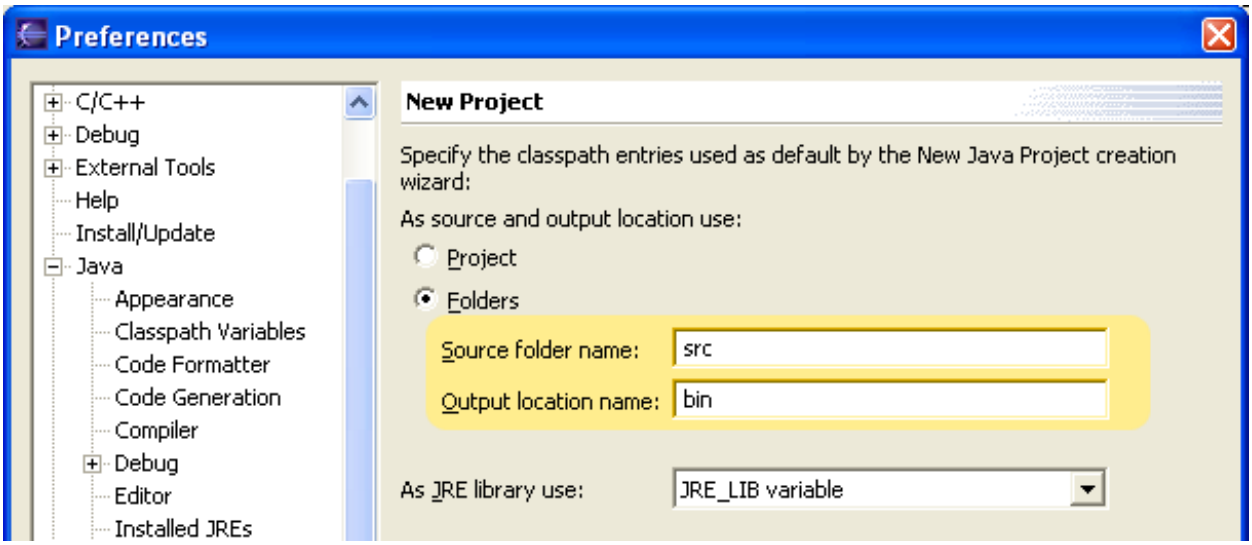
Further information on Eclipse can be found at Java WebApplications mit Eclipse .

[1] You can find a list of many Eclipse plugins at http://eclipse−plugins.2y.net/eclipse/index.jsp or in section 13.1.1

Chapter 3. Java projects with Eclipse

3.1. Basic adjustments

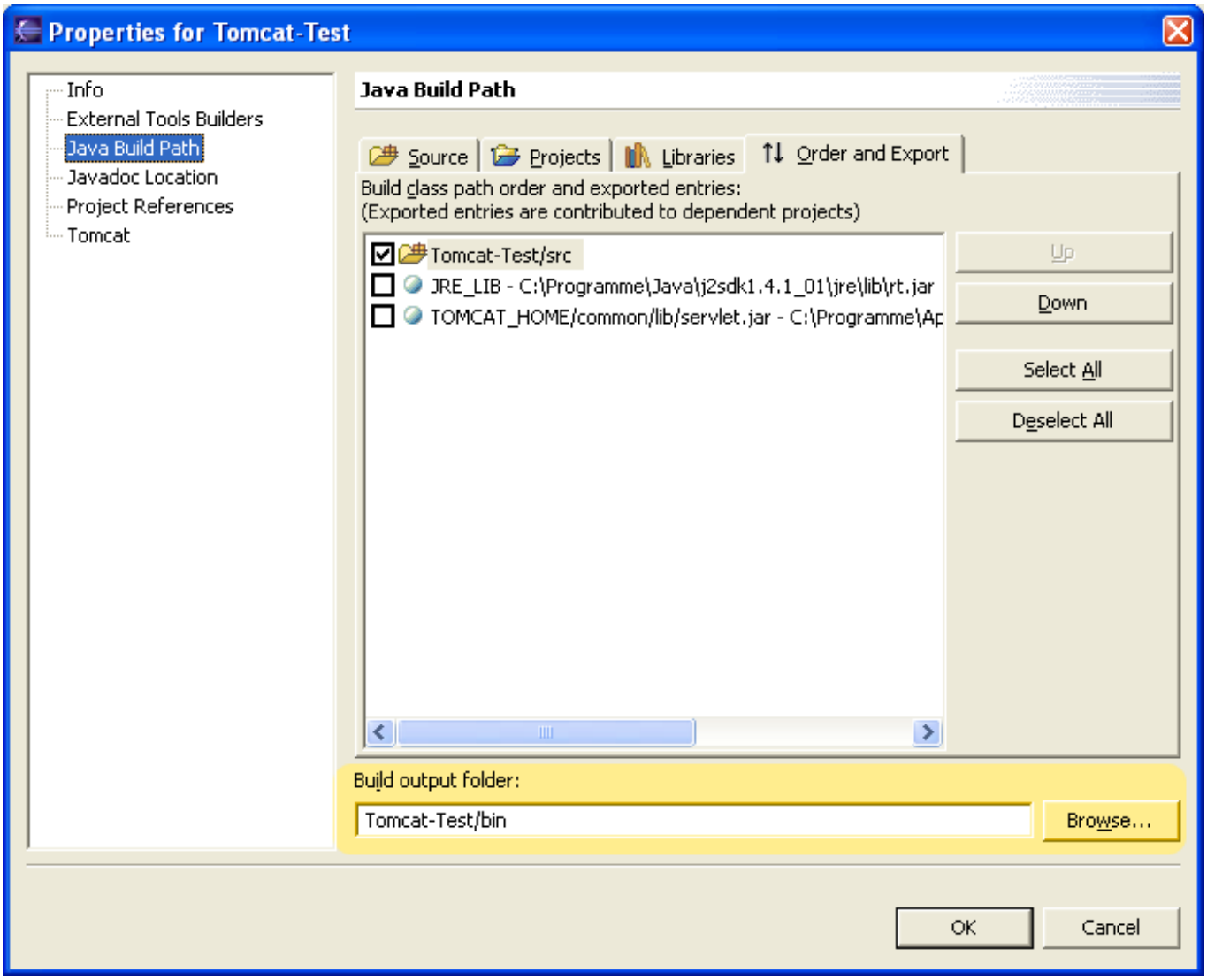
Figure 3.1. Preference New Project



Ensure that the source code remains separate from the produced class files. To do so, change the source folder name to src and the output location to bin in the dialogue Window − Preferences.

For web services it is even better to set the output directory to the path, where the application server expects them (`WEB-INF/classes`). You can also adjust this setting in Project−Preferences for each project separately.

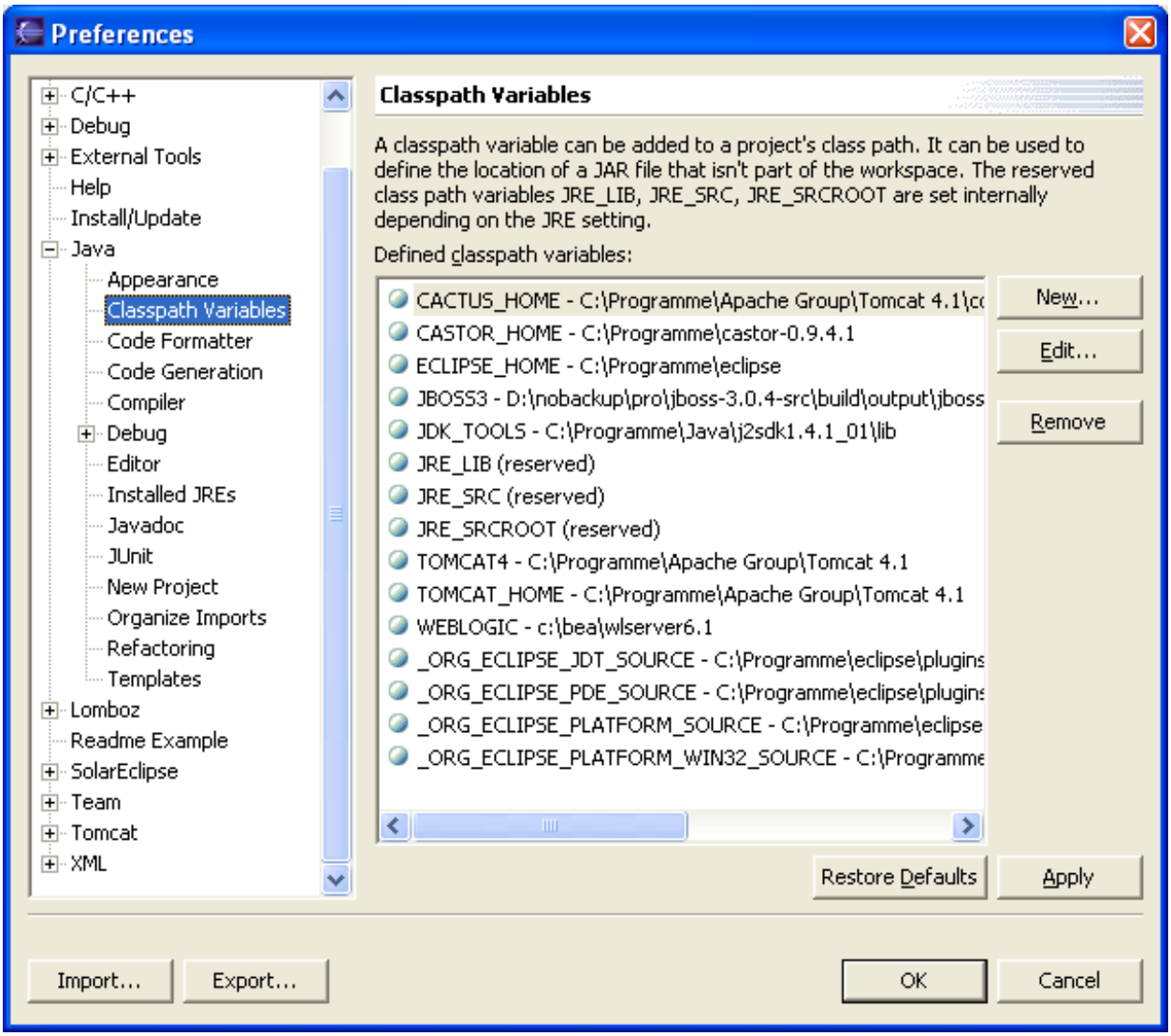
Figure 3.2. Properties Build output folder



3.1.1. Classpath Variables

Beside the Java runtime library you have to include other libraries in nearly every project. Since each developer might install them in different paths, you should use classpath variables.

Figure 3.3. Preferences Classpath Variables



In the project only symbolic names are stored. Each developer can adjust them to his local setting. If you need e.g. servlet.jar from Tomcat, it will be referenced in the project as TOMCAT_HOME/common/lib/servlet.jar . The actual installation path of Tomcat will be defined by each developer setting the value of classpath variable TOMCAT_HOME corrosponding to his local installation.

3.1.2. Name versioning

If you develop in a team and use relatively new technologies, tool versions and their libraries change rapidly. To avoid chaos and strange side−effects, which even look different for each team member, it is very important that everyone uses the same tool versions.

Therefore I recommend to use names with a version number in it e.g. castor−0.4.1.jar . If all libraries in the project are referenced this way, it is guaranteed that all developers use the same versions.

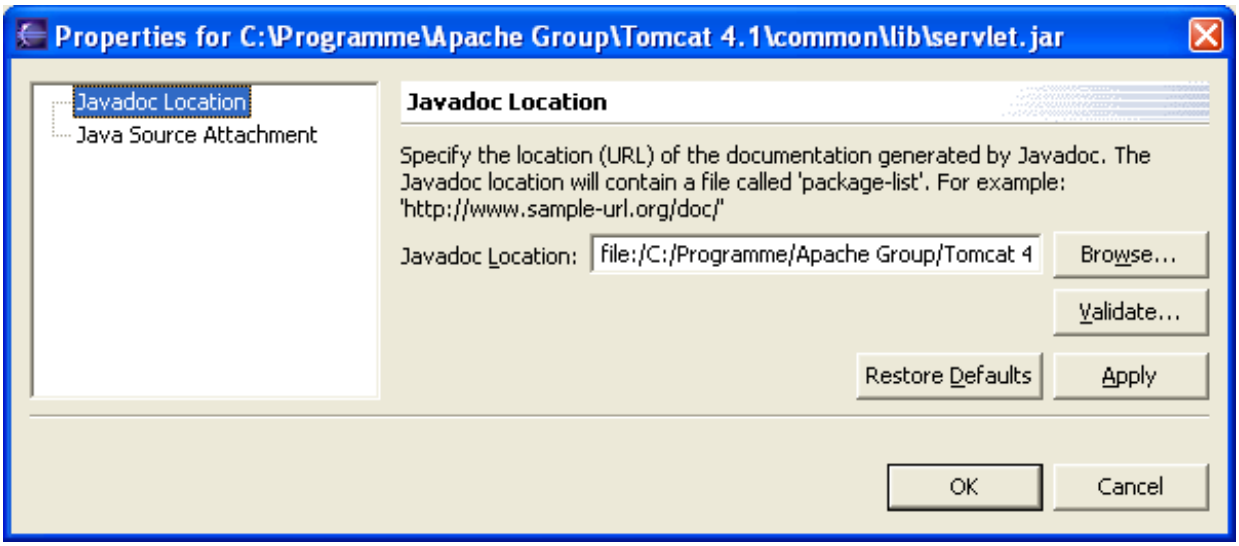
3.1.3. Extend the documentation

You can extend the included documentation very easily with your own texts. You can include all javadoc API documentation. This applies to Java sdk and additionally to a any library used in any project.

Plugins can also include their documentation smoothly − an example is this article which is available as documentation plugin in the download area. If installed, this article will be available and searchable in the Eclipse

help viewer.

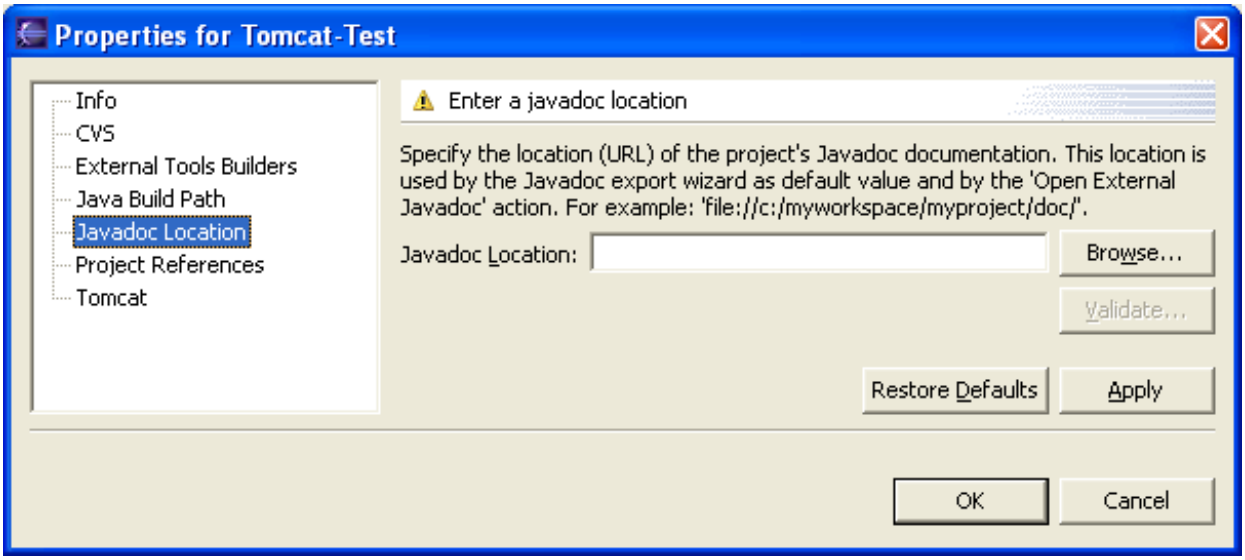
Figure 3.4. Properties for a library



To get most support you should assign the API documentation (javadoc) and even the source code (if avialable) to every library used in the project. If you install a complete java sdk, Eclipse will recognize this automatically. Assign the documentation and sourcecode archives in the Properties of each JAR (available in the context menu) for all individual libraries.

Accordingly the API documentation of your own project is included:

Figure 3.5. Properties include own documentation



Setup like this, press SHIFT + F2 to open the documentation for the current object or F3 to open the corresponding source file.

Chapter 4. WebServices

With Web Services, you can also realize distributed applications additionally to CORBA, DCOM or EJB. Web services define another standard for letting different software components work together over a network. Special attention was put on platform− independence, which means that web services can work together spanning different manufactures and systems. From the start, the Internet was considered as the connecting network with the goal to connect and integrate applications over the Internet.

Also keep in mind that Microsofts .NET Framework uses web services as integral technology for its distributed infrastructure. It uses web services instead of DCOM.

4.1. Web services messages

Web services messages actually always use the SOAP standard, which represents an XML application. Requests to a web service are thus expressed in XML. The answers returned by the web service are also delivered in XML. Later I will explain, how to look at such SOAP message for error tracing.

4.2. Web services transports

The transport of SOAP messages for web services nearly always takes place over http(s). Like mentioned before, the Internet is the preferred transportation network for web services and naturally http is used for transport. If using http, one can usually rely on a very good infrastructure, because http is commenly used in all other places for the www services. Accordingly you have fewer problems with firewalls and such things: http always works! .

4.3. WSDL web service definition language

A WebServices is defined or described in WSDL − a W3C standard . The description is written in XML and defines the interface web services similar to the interface definition language IDL of the OMG . The WSDL document mainly describes the methods offered by the web service, the parameters needed and what is returned.

4.4. Axis

The Axis framework of Apache (http://xml.apache.de/axis) supplies the necessary basis for providing web services to Java. With Axis, web services can easily be used , i.e. as client to call a web service and it alos offers extensive support for developing web services.

Axis cannot publish the web service itself, but runs as servlet in a servlet container such as Tomcat or JBOSS.

Chapter 5. XDoclet

For nearly all J2EE technologies, it is necessary for web services to deliver beside the actual java classes also a description (usually in XML), which describes attributes apart from the pure implementation.

These descriptions are called deployment descriptors, which are used at deployment time to define certain properties. This is the case for enterprise java beans und also for web services[2].

To avoid the problem changing multiple files, if e.g a classname changes, the attribute−oriented programming AOP with XDoclet uses the principle of including all information especially all attributes directly in the java source code. If anything changes, the metafiles will then be regenerated by XDoclet . By this all files are consistent and you have a better overview as you do not have to re−check at n places, what a class exactly does.

5.1. XDoclet comments

XDoclet uses javadoc comments, in order to accommodate the additional attributes in the java source code.

Example:

```
/**
 * @author sr
 * @axis.service name="MyTestService" scope="Request" enable−remote−admin="true"(1)
 */
public class TestService {
    ...
}
```

(1) Here a web service is defined by a XDoclet comment.

5.2. XDoclet templates

For producing the metafiles such as deployment descriptors, templates are used. XDoclet defines its own template language, which is exactly cut to its requirements. This example template prints the complete class name of all classes, which are defined as Axis Handler:

```
<?xml version="1.0" encoding="utf−8"?>
<!−− Test Template für XDoclet −−>
<XDtClass:forAllClasses>(1)
 <XDtClass:ifHasClassTag tagName="axis.handler" paramName="name">(2)
 <XDtClass:fullClassName/>(3)
 </XDtClass:ifHasClassTag>
</XDtClass:forAllClasses>
```

(1) Iterate over all classes, which are in the input quantity (the indicated file set).

(2) If a class has a class level tag with name axis.handler and a parameter name , then...

(3) print the complete classname.

All tags beginning with XDt belong to the template language of XDoclet and are descriped in XDoclet−documentation.

[2] The list can be extended at will: servlets, taglibs, OR mappings, ...

Chapter 6. Eclipse and J2EE

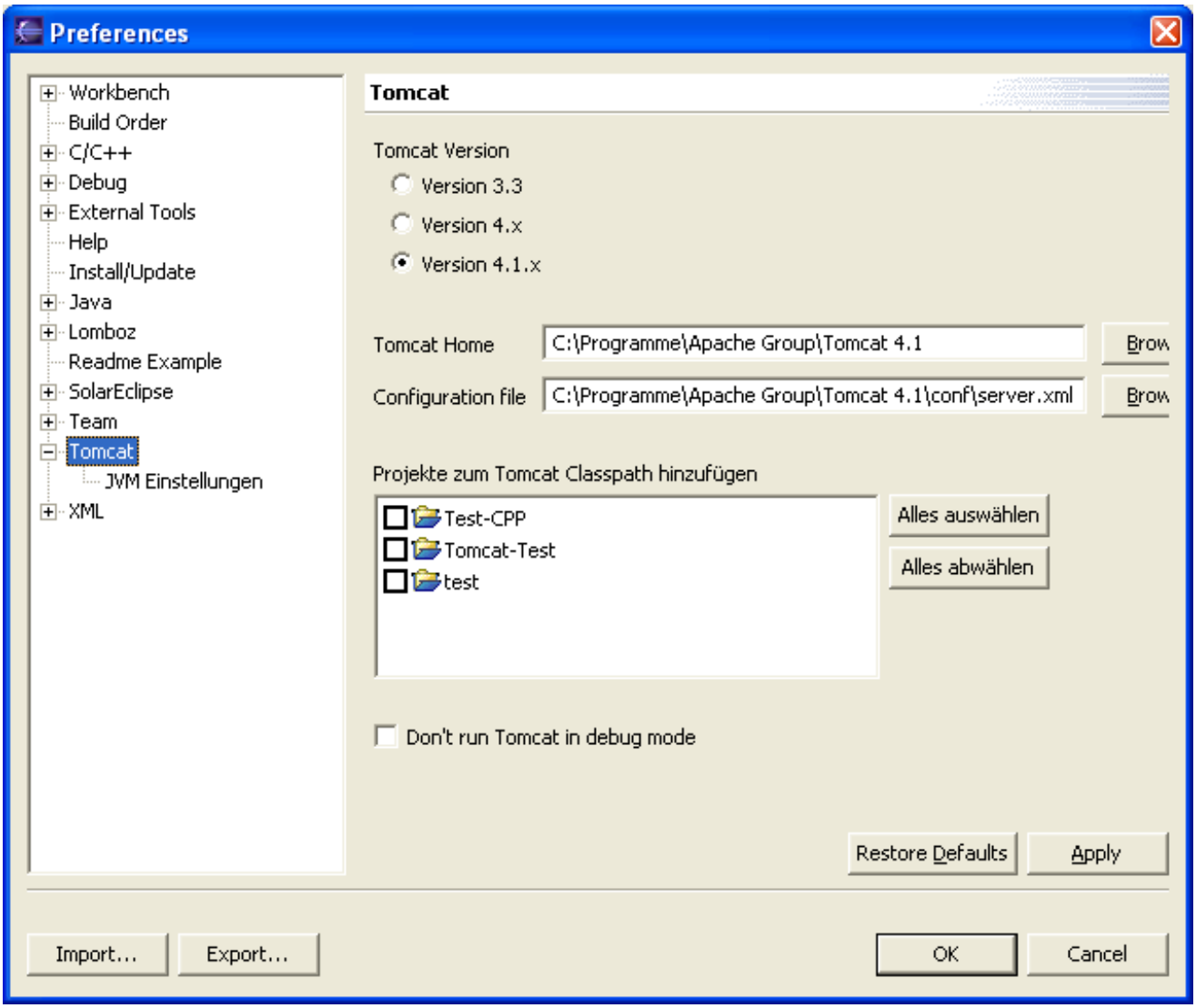
To develop web services with Eclipse a couple of plugins are recommended. The Tomcat plugin of Sysdeo is well suitable for simple projects with Tomcat as application server. Since I would like to develop and debug web services with Tomcat as container later, it is used in this project. Substantially however the Lomboz Plugin of ObjectLearn supplies more functionality. In addition, it is primarily helpful for developing EJB applications, but also has a few functions for web services.

Optionally, the plugin JBOSS IDE can be used, in order to steer XDoclet more comfortably. I will present both kinds manually (by ant script) and with JBOSS IDE.

6.1. Tomcat plugin from Sysdeo

First unpack the zip archiv in the directory hierarchy. After restarting Eclipse you can configure the plugin in Windows / Preferences :

Figure 6.1. configure the tomcat plugin

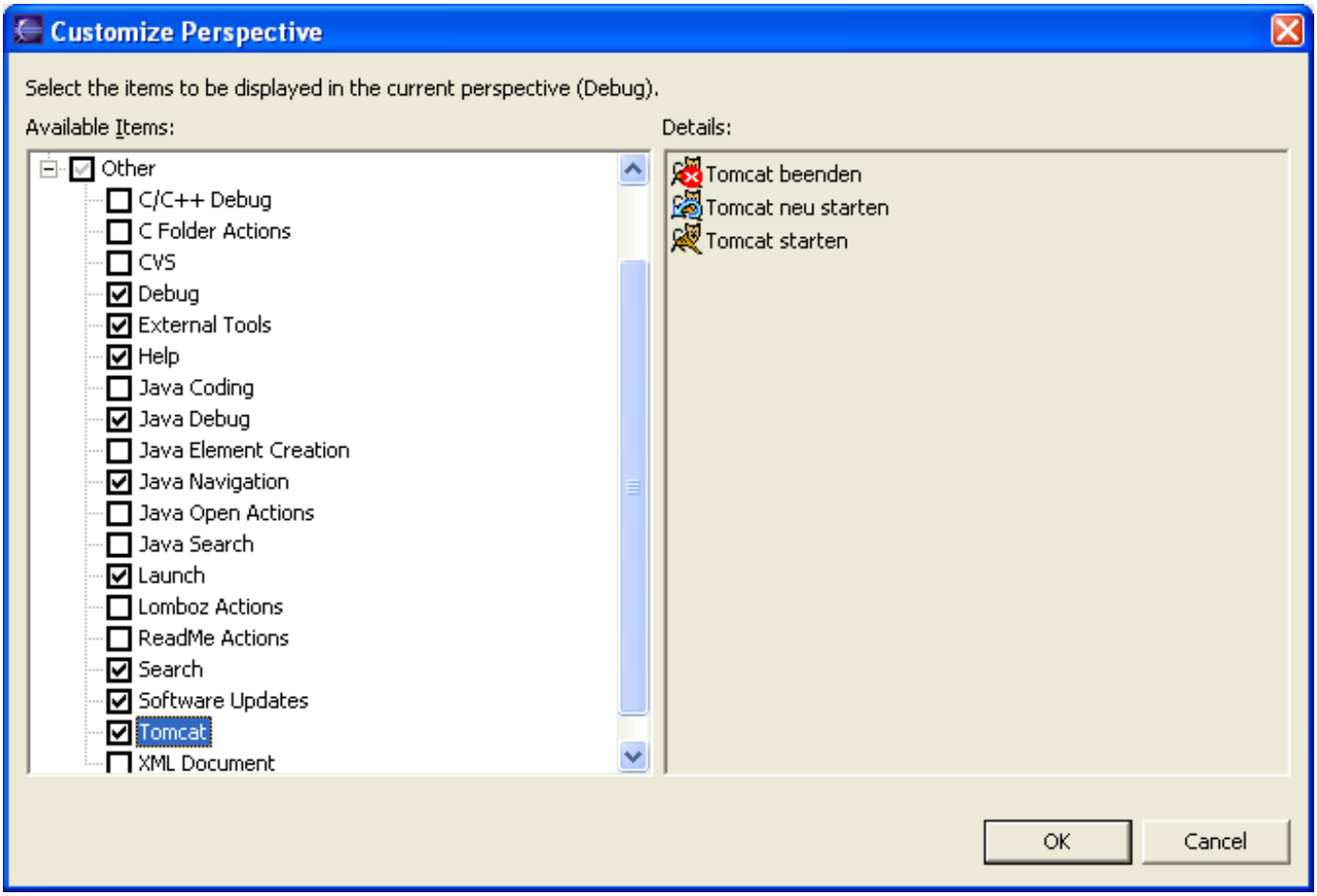


The most important properties are:

- Tomcat version
- Tomcat home directory
- Tomcat config file (server.xml) to use

Thus the Plugin can be used. In Windows/Customize Perspective one activates the plugin in the Debug−Perspective :

Figure 6.2. activate tomcat plugin



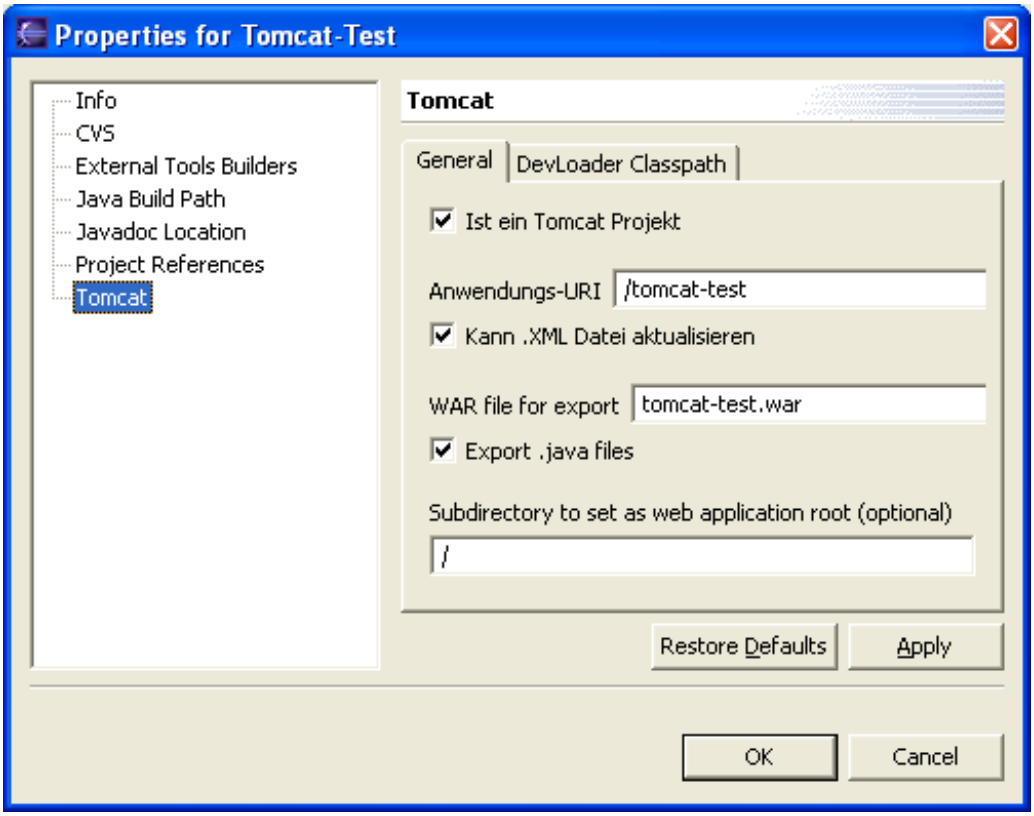
Three new buttons  are added to toolbar with which Tomcat can be start and stop directly from Eclipse.

The interesting thing is, that you can debug a web service including Tomcat in Eclipse by placing breakpoints in a doGet method of a servlet or in a web service method.

6.1.1. Project as Tomcat project

From now on you can choose Tomcat project , when creating a new project. An existing project can be change to Tomcat project in Project−Properties :

Figure 6.3. tomcat project properties

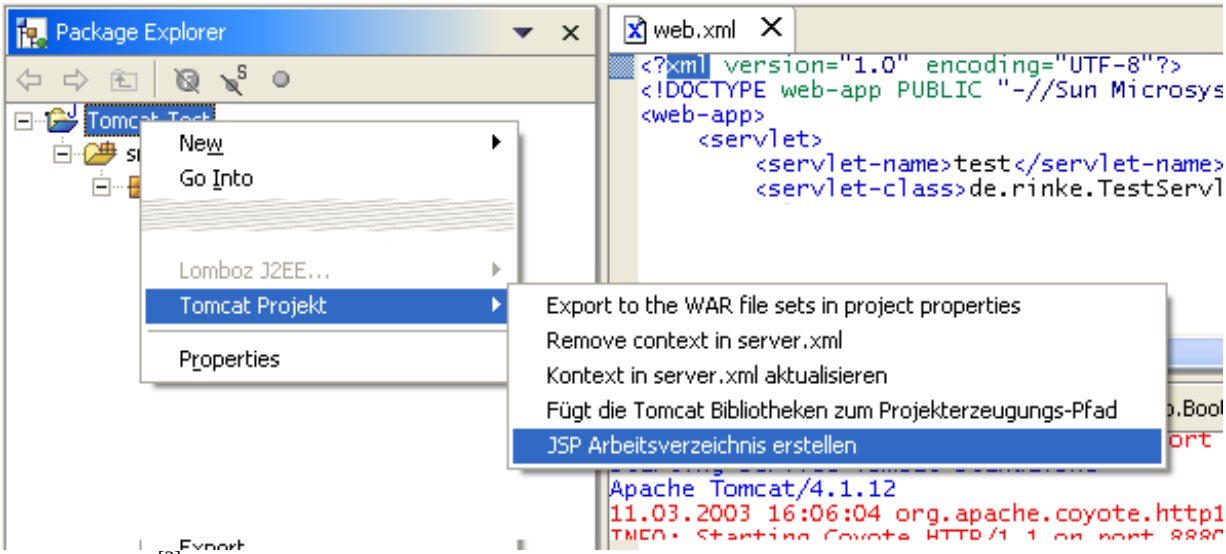


- Check Is a tomcat project .

- Application URI must contain the context path for Tomcat.
- If can actualise XML is checked, the Tomcat context will be updated in the server.xml automatically.
- WAR file for export sets the name of the war file, when the project is exported. The name is an absolute pathname, referencing directly to the directory where the war file is deployed.

If everything is set up like this, then the context menu of the project has an additional entry :

- **Figure 6.4. tomcat project options in context menu**



Export WAR[3] file.

- The context in the Tomcat configuration file (server.xml) can be removed and/or updated.
- The Tomcat libraries can be added to the project.
- Create a work directory, which takes the generated servlets from JSP pages.

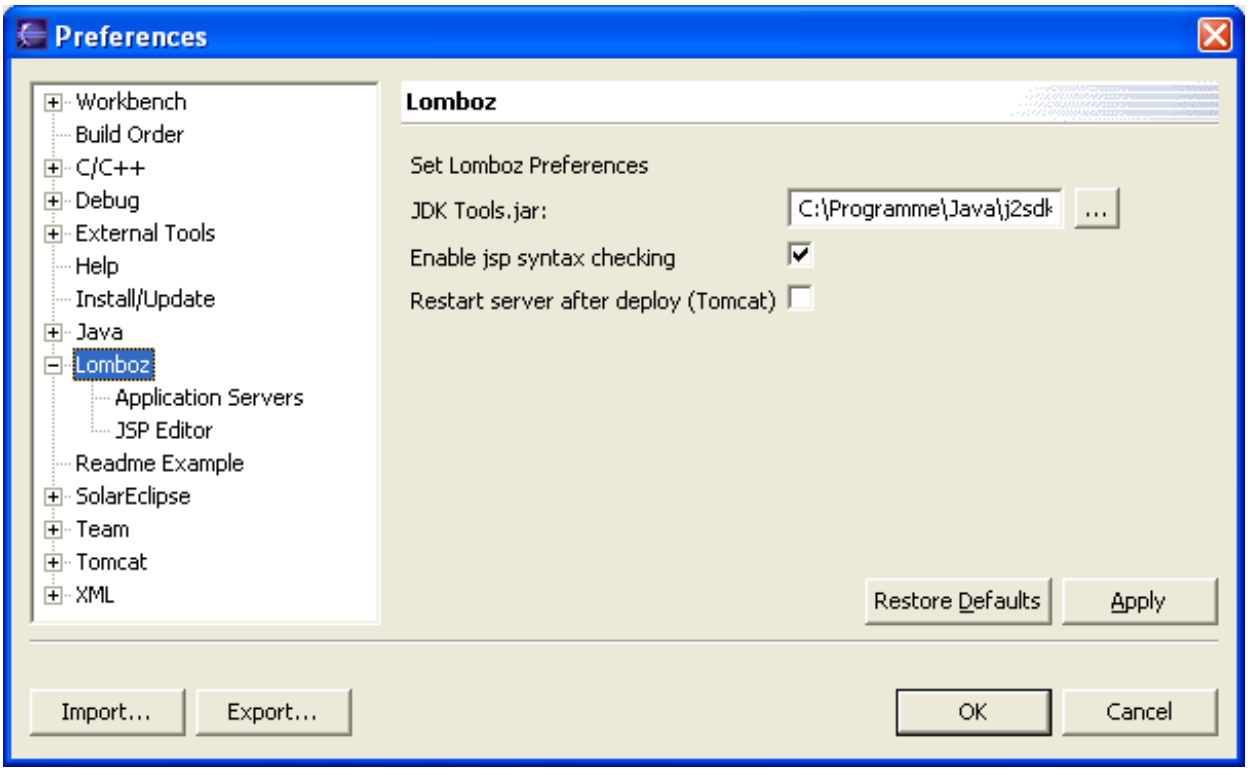
6.2. Lomboz plugin from ObjectLearn

ObjectLearn created a very powerful tool with its plugin for developing J2EE applications. This article only introduces only a small portion of the entire functionality covering web services. The whole EJB specific part and the most servlets information is left out.

6.2.1. Installing and set−up

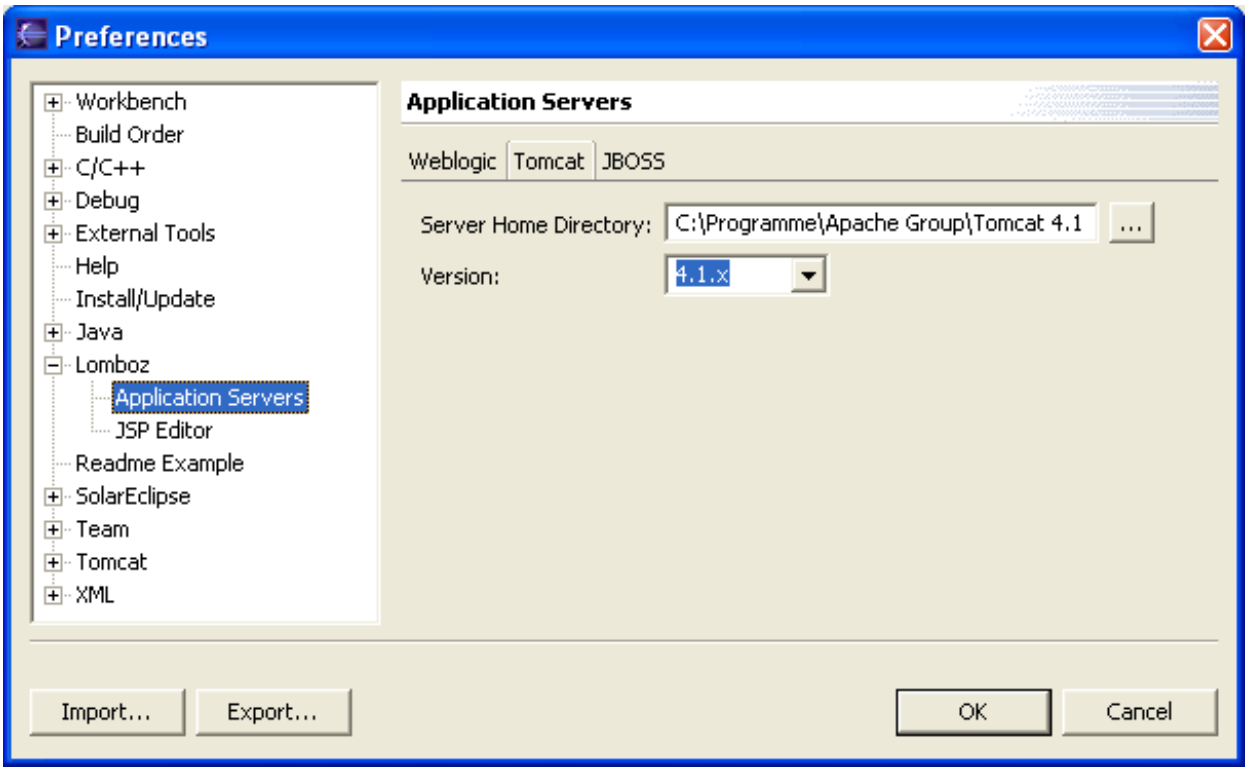
First unpack the zip archiv in the directory hierarchy. After restarting Eclipse, you can configure the plugin in Windows / Preferences :

Figure 6.5. configuration for Lomboz 1



Set the path to tools.jar . This is the library from the JDK which contains the compiler. If you have installed several JDKs, make sure to use the same the compiler of the JDKs as is used for the project.

Figure 6.6. configuration for Lomboz 2

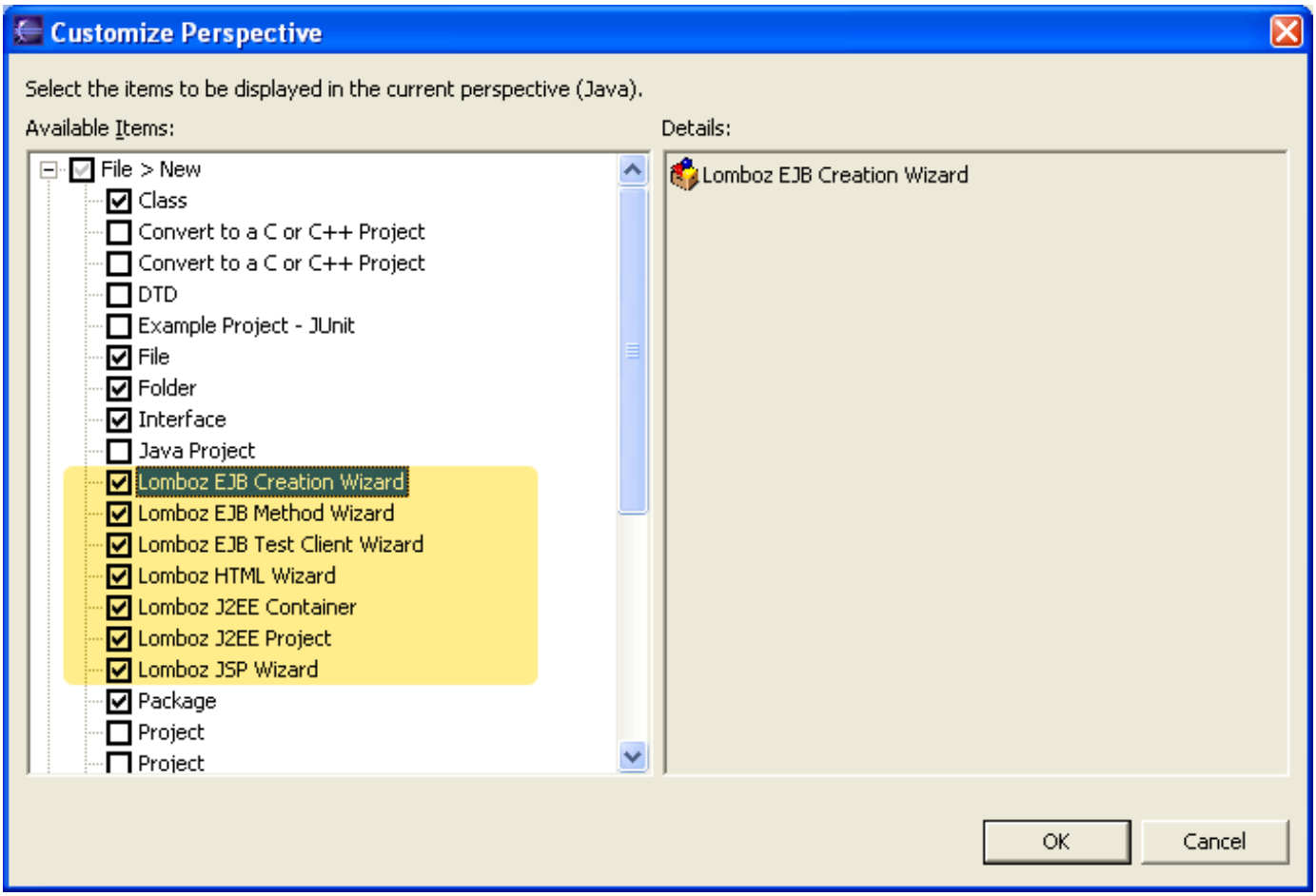


You can then select the suitable parameters for your Tomcat installation.

The attitudes JSP editors are rather of cosmetic nature and are not described in more detail.

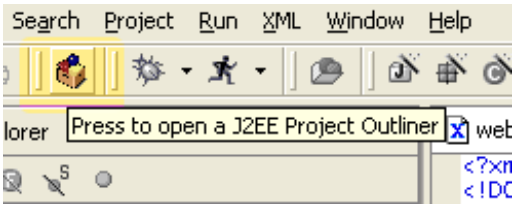
To make the different views and actions visible, it is nessesary to activate some options in Customize Perspective in the java perspective (same as for the Tomcat plugin):

Figure 6.7. configuration for Lomboz 3



Accordingly you must activate the Lomboz Actions in the node Windows / Show View the Lomboz J2EE View and in node Other.

Beside the wizards for creating new resources, a new button is added to the toolbar of the J2EE Project Outliner .



6.3. Wizards of the Lomboz plugin

Apart from the administration of J2EE containers, Lomboz has more functionality to offer. There are wizards for producing various objects. Except the EJB stuff there are the following:

- Lomboz J2EE project: produces the complete project structure in one shot, the result rather looks exactly the same, as which is introduced in the following
- Lomboz HTML page: produces a skeleton for a HTML page
- Lomboz J2EE container: produces a new J2EE (Web) container (as described above)
- Lomboz JSP page: produces a skeleton of an new JSP page. Beside the definition of the error page, you can define the beans to use with id, scope and so on.
- Lomboz servlet wizard: produces a skeleton of a servlet class.
- Lomboz SOAP client cizard: produces a new stub class for accessing a web service

Except the SOAP client wizard nothing will be used here.

[3] (W)eb−(A)pplication−A(R)chive

Chapter 7. Preparations

The actual Axis installation runs as described in http://ws.apache.org/axis/java/install.html. In order to optimize testing and debugging, a different setup is used.

The project is created as a Tomcat project (a web application) in Eclipse and the runtime libraries of Axis are integrated. With this setup the whole application runs inside Eclipse which is best for debugging. You can stop any request at any time by simply placing a breakpoint in the web service methods. The output directory is set to the classpath of the web container (I will use Tomcat), so any update or change to a Java class will immediately reflected by the running application. Any copying or redeploying can be omitted. Tomcat recognizes any change and reloads the related class automatically.

The following steps are required:

- create a J2EE project with Lomboz
- create a web container
- use Sysdeo plugin to start the whole application directly in eclipse
- include Axis runtime libraries

7.1. Configure plugins

The plugins from Sysdeo and ObjectLearn should be installed by unzipping the archives into the Eclipse plugin directory. Adjust a few paths in both plugins. Because I will use Tomcat as application server, the Tomcat path in both plugins must be set to your Tomcat installation (for windows c:/Programme/java/tomcat−4.1.29 for example).

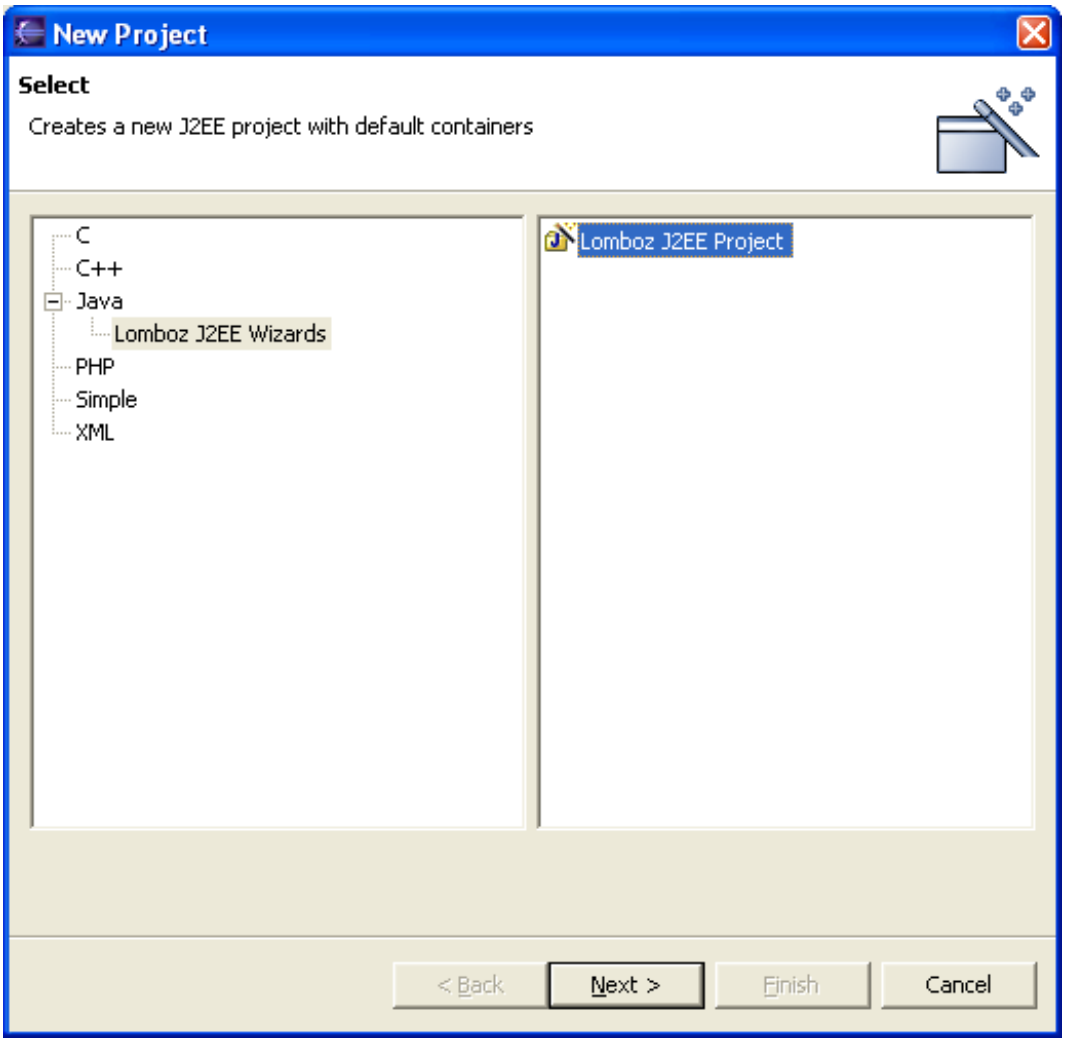
The Lomboz plugin needs a few boxes checked to activate all wizards and the other stuff (see chapter 6.2).

Please note: the used Tomcat version 4.1.29 must be treated as Tomcat 5.X in the sysdeo plugin, otherwise starting the server will result in some weird errors.

7.2. Create a J2EE project and a web container

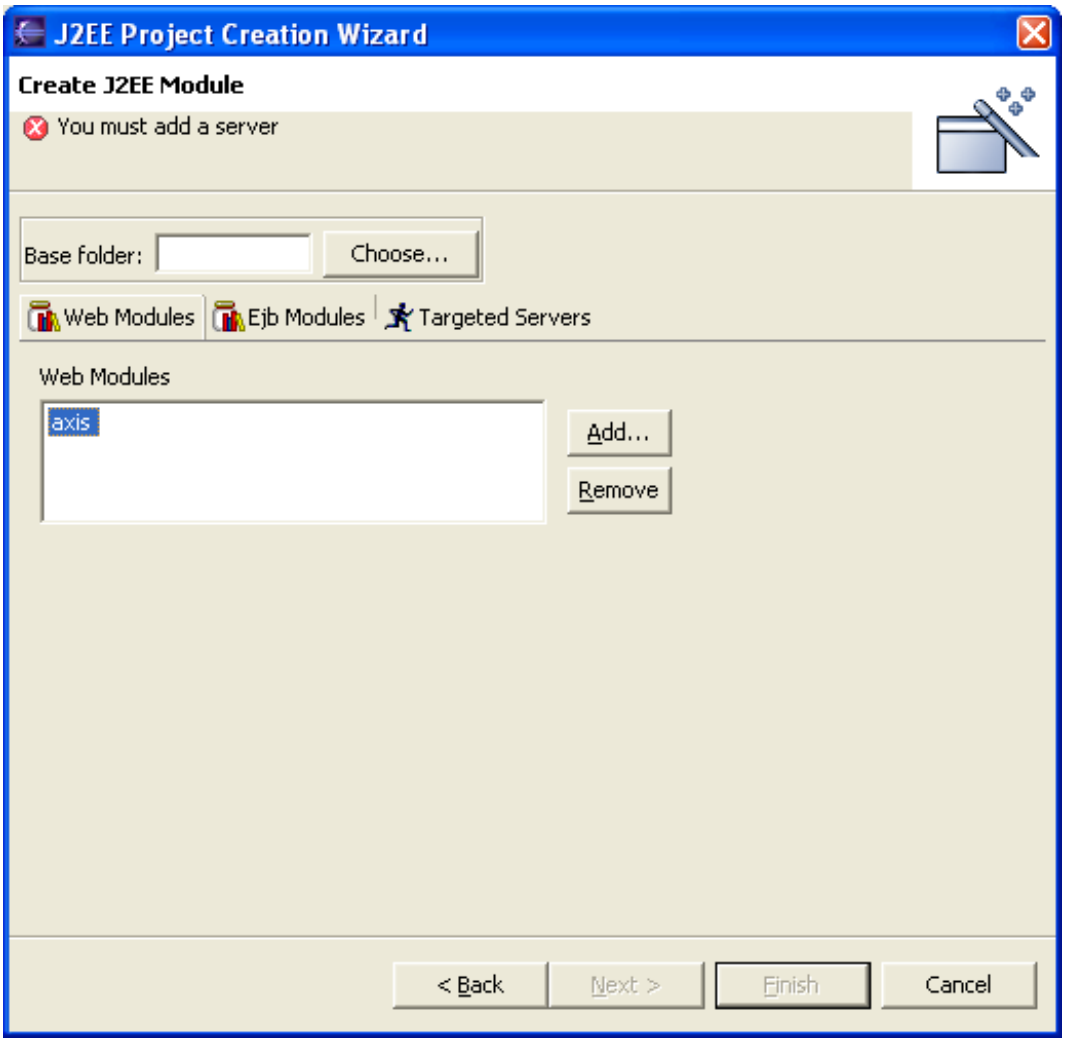
Use New > Project: Java > J2EE Project to create a new project.

Figure 7.1. create a new project with lomboz



After setting the name and all java specific stuff you must add a new web container with Tomcat 4.1.x as target server.

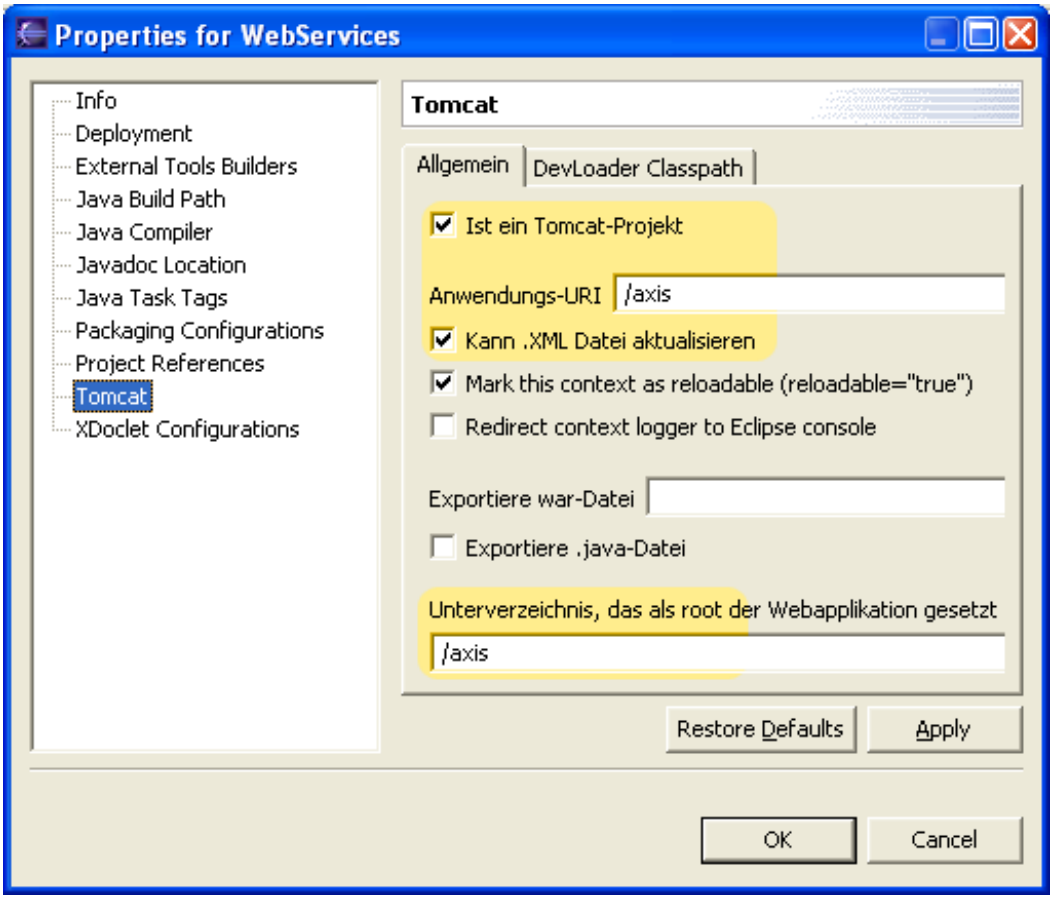
Figure 7.2. create a web module with Lomboz



7.3. Start web application in eclipse

In project properties you check Is a Tomcat project and set the path like shown below. Thus Tomcat can be started directly in Eclipse and the web application runs in the Eclipse workspace. No more copying of class files is required.

Figure 7.3. project properties for Sysdeo tomcat plugin



The application URI determines the uri prefix for the browser URL (here http://localhost:8080/axis/). The sub−directory web application root (here also set to axis) determines the project sub−directory for the web container.

7.4. Intergrate Axis

The Axis installation overlaps somewhat with the Lomboz plugin, which already contains an Axis runtime installation. However this runtime is not the same as the current Axis version and not all libraries are included. In addition, Lomboz sets the classpath variable AXIS to the included Axis runtime.

To overcome all these problems, you should simply copy the actual Axis libraries to the Axis runtime that comes with Lomboz (copy AXIS_BASE/lib/*.jar ECLIPSE_BASE/plugins/com.objectlearn.jdt.j2ee/axis). Be careful as this does always works, but with the specified versions, there are no problems.

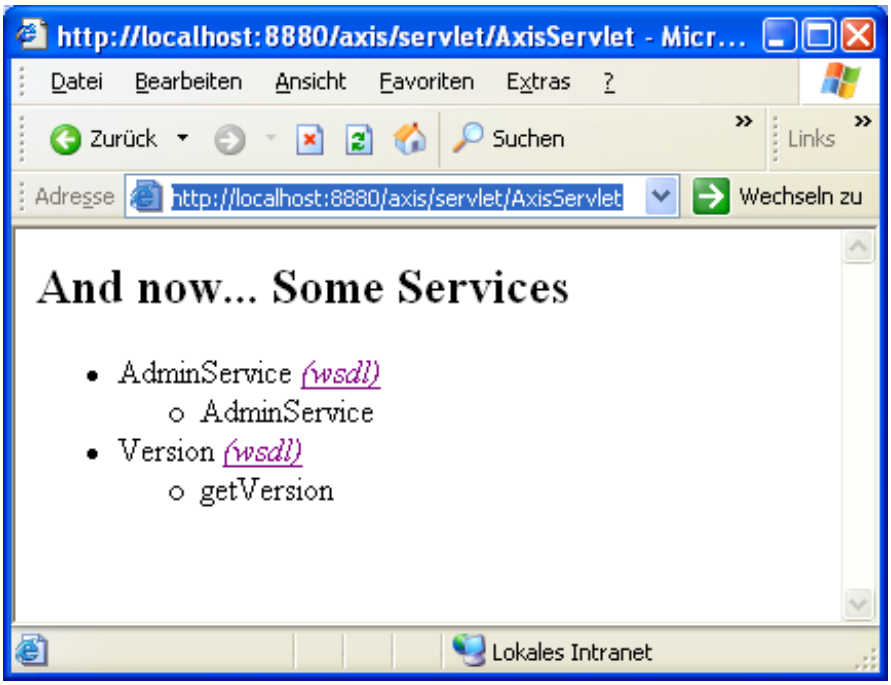
7.5. Insert Axis web application

Copy the Axis sample web application in the project workspace (or import the files with Eclipse): (copy AXIS_BASE/webapps/* WORKSPACE/<project name>/axis). The class files, which are also included in the binary distribution of Axis, are not needed and can be deleted (placed in WEB−INF/classes).

7.6. First test

Start Tomcat with the sysdeo plugin and access the Axis sample application with http://localhost:8080/axis/servlet/AxisServlet. You should see already two web services:

Figure 7.4. list of all deployed web services

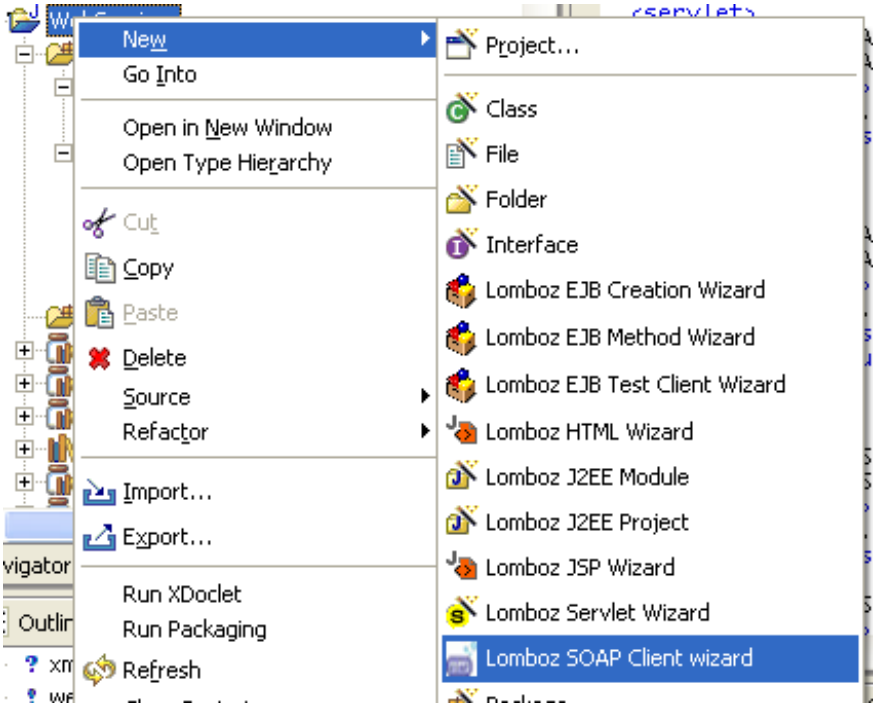


Click on the wsdl −link to have a look at the wsdl description of these services.

Chapter 8. Use web services

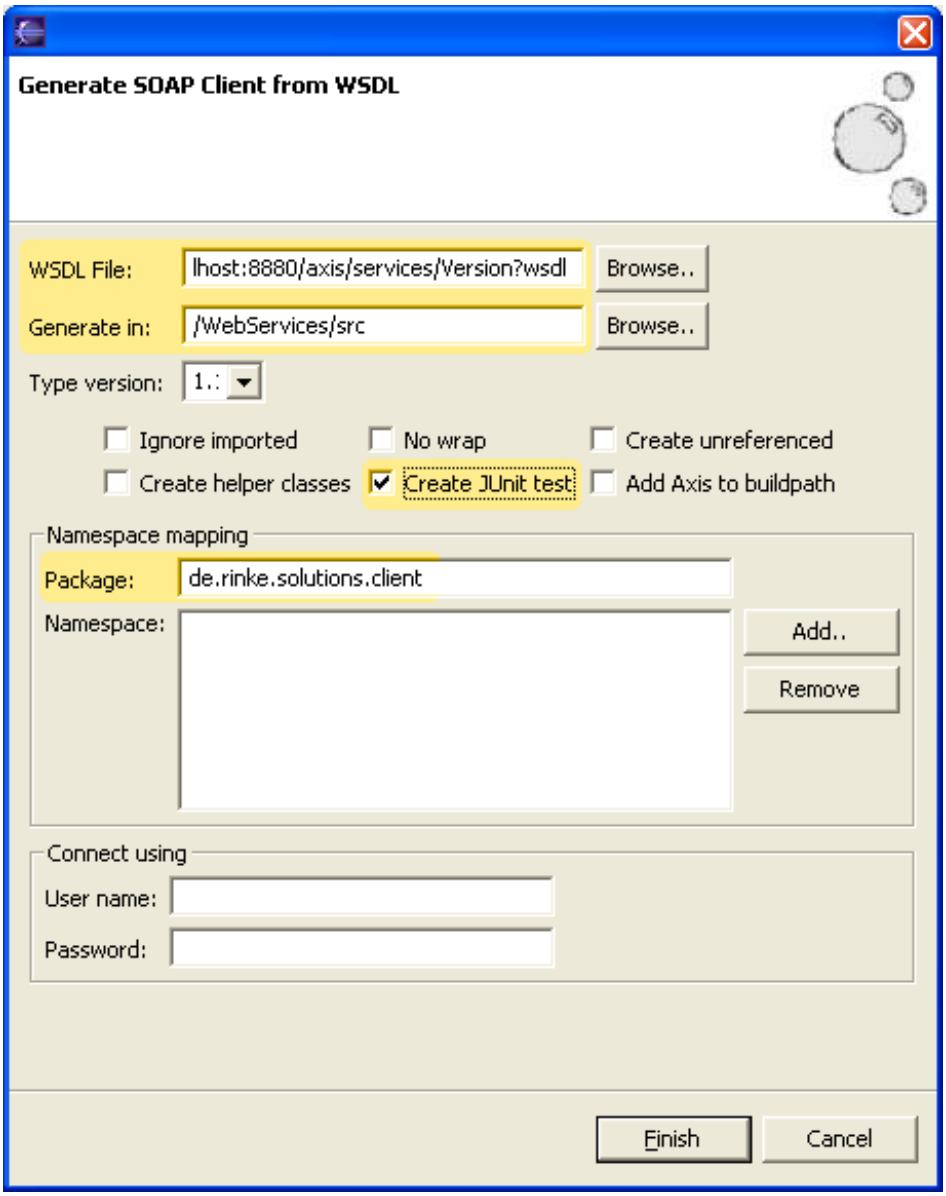
There are several possibilites to access web services from Java. The easiest way is to generate a client stub directly from the wsdl description. This stub represents the web service on the client side. You can use this stub exactly like every other java class, the fact that a web service is working behind the scenes is not noticeable[4]. The Axis framework comes with the tool called WSDL2Java for generating those stubs. Since direct calling is nevertheless rather pedantic, we let that to the Lomboz plugin:

Figure 8.1. Call the web service wizards of Lomboz



You can either read the wsdl file from the filesystem and save the file with the browser. Or you can alternatively use the url http://localhost:8080/axis/services/Version?wsdl directly:

Figure 8.2. web service wizard in lomboz



If you check create Unittest , Lomboz will also generate a unit test testing the web service. This is a simple way to show how to use the generated classes. The test class requires junit.jar to be added to the project libraries because it makes use of it.

Subsequently there are four new source files in the indicated package, that together enable the access to the version web service:

- Version.java is the interface descriping the web service.
- VersionService.java is the interface for creating a stub (the factory).
- VersionSoapBindingStub.java is the implementation of the Version interface.
- VersionServiceLocator.java is the implementation of the factory.

The fifth file is the test case, which is calling the web service. To run this test case, select the resource VersionServiceTestCase.java and choose Run / Run as ... / JUnit TestCase from the menu bar.

---

[4] This is not completely correct, since by the remote procedure call certain exceptions cannot be avoided.

Chapter 9. Create web services

9.1. Web services with JWS files

Here is the easiest way to create a web service with Axis: Take a simple java class, which has one or more public methods and rename the *.java file in *.jws.

Like JSP compiled to servlets, Axis will compile these jws files at the first access into a complete web service. An explicit deployment is not necessary.

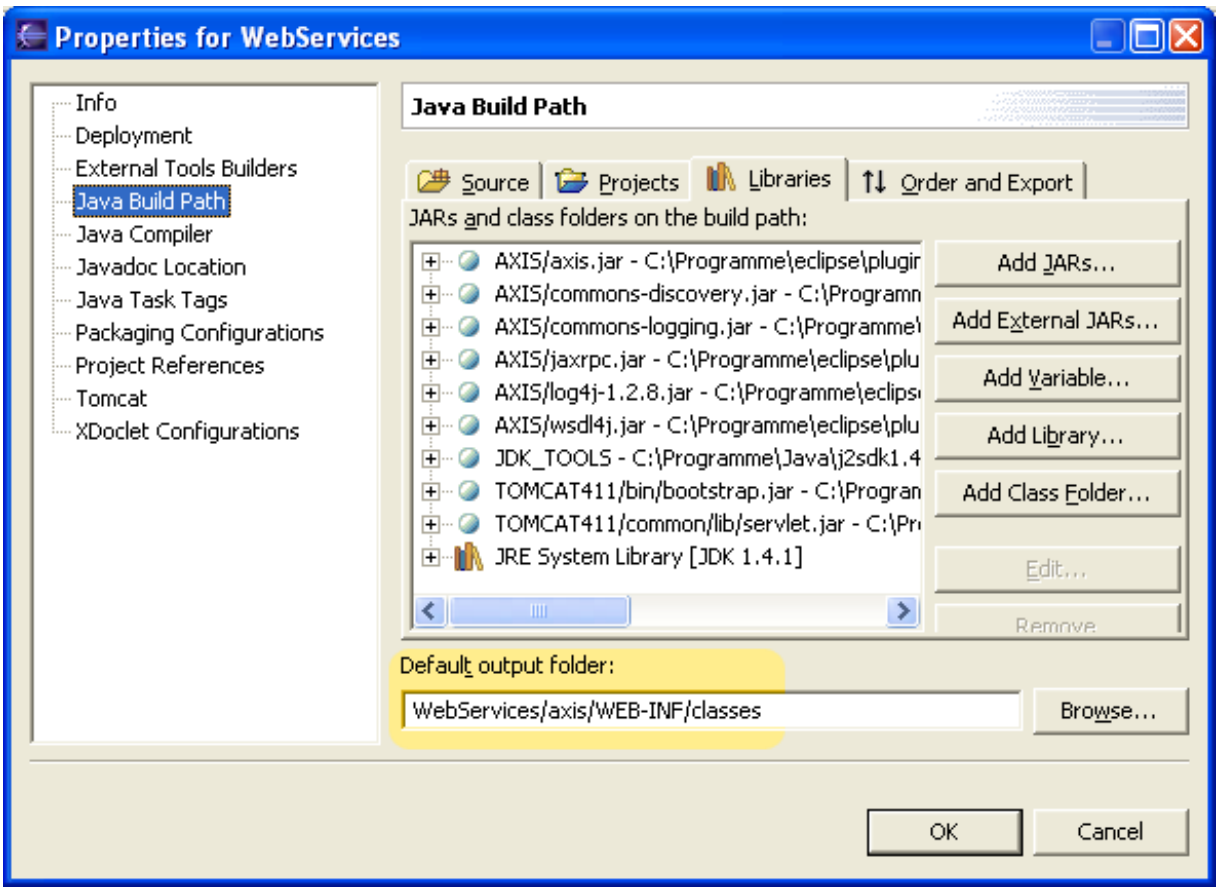
The file EchoHeaders.jws which is included in the Axis distrubution, is a sample for such a web service. The compiled class file is located in WEB−INF/jwsClasses. If you want to debug such web services some dodges are nessesary. In addition, the control via a deployment descriptor is not intended here.

9.2. Web services as normal java classes

You could implement a web service as a simple java class (plain old java object or POJO). All you need is an additional deployment descriptor. To avoid maintaining this descriptor seperately from the java source code, I will use XDoclet comments to generate it.

Beside the actual java class you need two more things: an ant script and a XDoclet template.

Figure 9.1. change the output folder



To force Eclipse to write the compiled class files to the right location, set the output directory in project preferences directly to the web containers WEB−INF/classes directory (see example project):

9.2.1. A web service

In the included sample project an easy web service is realized:

```
/**
 * @author sr
 * @axis.service name="MyTestService" scope="Request" enable−remote−admin="true"(1)
 * @axis.useHandler name="track"
 */
public class TestService {

    /**
     * calcs a sum.
     * @axis.method(2)
     * @param p1 first parameter
     * @param p2 second parameter
     * @return the sum of both parameters
     */
    public int sum( int p1, int p2 ) {
        return p1+p2;
    }

    /**
     * gets the address
     * @axis.method
     * @param id
     * @return the address
     */
    public String getAddress( int id){
        return null;
    }

}
```

(1) these XDoclet tags declare the class a web service.

(2) @axis.method marks a method for export as web service method. This method is thereby callable extermally .

9.2.2. The XDoclet template

Included in the sample project is a special XDoclet template, which generates a deployment descriptor for Axis. It evaluates the tags described in chapter 10 and generates a deploy.wsdd file.

```
<?xml version="1.0" encoding="utf−8"?>

<deployment
 xmlns="http://xml.apache.org/axis/wsdd/"
 xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
 xmlns:xsi="http://www.w3.org/2000/10/XMLSchema−instance">

<XDtClass:forAllClasses>

<!−− ... check for Handlers −−>

</XDtClass:forAllClasses>

<XDtClass:forAllClasses>
```

```
<XDtClass:ifHasClassTag tagName="axis.service" paramName="name">

 <service name="<XDtClass:classTagValue tagName="axis.service" paramName="name"/>"

<!-- ... -->

     <XDtClass:ifDoesntHaveClassTag tagName="axis.service" paramName="provider">
       <XDtType:ifIsOfType type="javax.ejb.EntityBean,javax.ejb.SessionBean">
       provider="java:EJB"
       </XDtType:ifIsOfType>
       <XDtType:ifIsNotOfType type="javax.ejb.EntityBean,javax.ejb.SessionBean">
       provider="java:RPC"
       </XDtType:ifIsNotOfType>
       >
     </XDtClass:ifDoesntHaveClassTag>

     <XDtClass:ifHasClassTag tagName="axis.useHandler" paramName="name">
     <requestFlow>
         <handler type="<XDtClass:classTagValue tagName="axis.useHandler" paramName="name"/>"/>
     </requestFlow>
     </XDtClass:ifHasClassTag>

     <parameter name="className" value="<XDtClass:fullClassName/>" />

    <!-- check if remoteAdmin param is present -->
     <XDtClass:ifHasClassTag tagName="axis.service" paramName="enable-remote-admin">
       <parameter name="enableRemoteAdmin" value="<XDtClass:classTagValue tagName="axis.service" param
     </XDtClass:ifHasClassTag>

     <parameter name="allowedMethods"

     <XDtClass:ifDoesntHaveClassTag tagName="axis.service" paramName="include-all">
       <XDtEjbSession:ifStatelessSession>
                value="create"
       </XDtEjbSession:ifStatelessSession>

       <XdtEjbSession:ifStatefulSession>
                value="<XDtMethod:forAllMethods><XDtEjbHome:ifIsCreateMethod><XDtMethod:methodName/> <
       </XDtEjbSession:ifStatefulSession>

       <XDtType:ifIsNotOfType type="javax.ejb.SessionBean">
                value="<XDtMethod:forAllMethods><XDtMethod:ifHasMethodTag tagName="axis.method"><XDtMe
 </XDtMethod:ifHasMethodTag></XDtMethod:forAllMethods>"
       </XDtType:ifIsNotOfType>

     </XDtClass:ifDoesntHaveClassTag>

     <XDtClass:ifHasClassTag tagName="axis.service" paramName="include-all">
                value="*"
     </XDtClass:ifHasClassTag>
     />

     <parameter name="scope" value="<XDtClass:classTagValue tagName='axis.service' paramName='scope' v

 </service>

</XDtClass:ifHasClassTag>

</XDtClass:forAllClasses>

</deployment>
```

9.2.3. The build script

The ant script controls the generation of the deployment descriptor with XDoclet and can afterwards deploy the web service into the running Axis. To achieve this, include Axis ant tasks contained in axis−ant.jar and deploy the web service with the admin task . The script is configured in build.properties, which contains all local pathes and settings.

```
<project name="webmodulebuilder" default="axis−deploy" basedir=".">

 <!−− set global properties for this build −−>
 <property file="build.properties"/>
 <property name="dist" value="../../dist" />
 <property name="web" value="../" />

 <property name="src" value="../../src/" />
 <property name="output" value="${basedir}" />

 <path id="XDoclet.class.path">(1)
    <fileset dir="${XDocletlib.dir}">
        <include name="*.jar"/>
    </fileset>
 </path>

 <path id="axis.classpath">(2)
     <!−− use the webapps lib path, where all axis jars are present −−>
     <fileset dir="${basedir}/lib">
         <include name="**/*.jar" />
     </fileset>
 </path>

 <target name="axisdoclet">
         <taskdef(3)
             name="templatedoclet"
             classname="XDoclet.DocletTask"
             classpathref="XDoclet.class.path"
         />

         <tstamp>
             <format property="TODAY" pattern="d−MM−yy"/>
         </tstamp>

         <templatedoclet destdir="${output}" verbose="1">(4)
            <fileset dir="${src}">
                <include name="**/*.java" />(5)
            </fileset>

            <template templateFile="axis−wsdd.xdt.xml" destinationFile="deploy.wsdd">
                <configParam name="Xmlencoding" value="utf−8" />
            </template>

         </templatedoclet>
 </target>

 <target name="axis−deploy" depends="axisdoclet">(6)

     <taskdef resource="axis−tasks.properties" classpathref="axis.classpath" />(7)

     <axis−admin(8)
         port="${target.port}"
         hostname="${target.server}"
         failonerror="true"
```

```
          servletpath="${target.appname}/services/AdminService"
         debug="true"
          xmlfile="deploy.wsdd"
     />

 </target>
</project>
```

(1) This path definition sets the classpath for XDoclet. To use XDoclet tasks in ant the XDoclet libraries must be available. Thus build.property XDocletlib.dir sets the directory where to find the libraries.

(2) This path definition sets the classpath for Axis. To use Axis tasks the Axis libraries must be available. In this example I will use a reference to the WEB−INF/lib directory, because all Axis libraries are already installed in lib.

(3) This taskdef tag defines the template task for XDoclet.

(4) The templatedoclet tag excutes the XDoclet template to generate the deployment descriptor.

(5) The fileset specifies that all java source files in the src folder are taken into account.

(6) The axis−deploy target executes the deployment. It depends on the axisdoclet target, so the deployment descriptor is generated if nessesary. Note: the axis application must run inside Tomcat to deploy a new web service.

(7) This taskdef tag defines all ant tasks supported by ant (see Axis documentation).

(8) The axis−admin task actually deploys the web service. For this all parameters in the build.property file must sets to their correct values (especially target.port, target.server and so on). In addition the server must run and the AdminServlet must be activated (see web.xml in the example project).

You can see the active web services by rerequesting the Axis page /axis/servlet/AxisServlet. By the way the servlet itself uses the xml file server−config.wsdd to list all active web services. When a new service is deployed Axis will adjust this file accordingly.

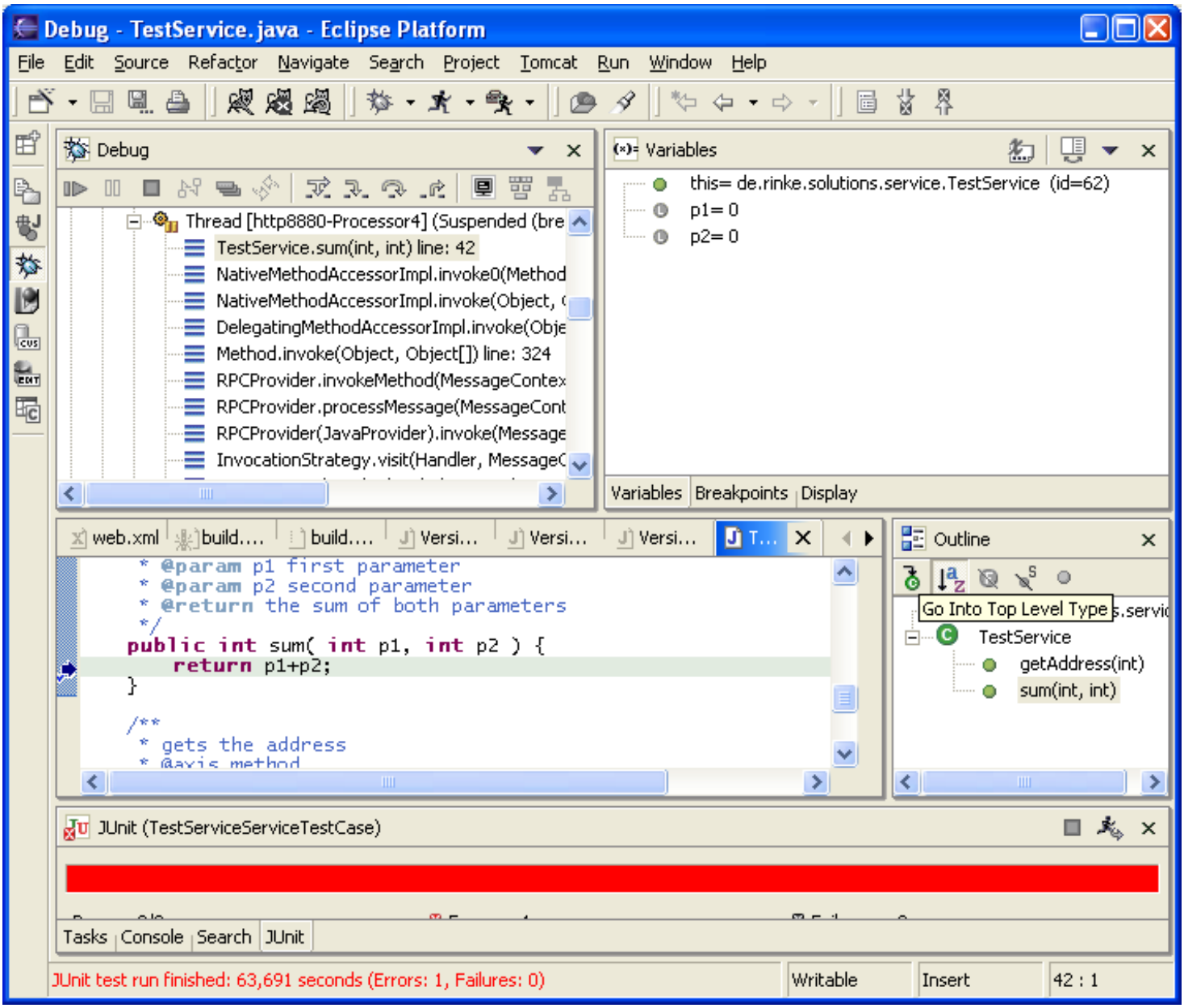
9.3. Web services as Enterprise Java Beans

I will provide this chapter in a later version.

9.4. Debug web services

Because the whole web service including Tomcat is running inside Eclipse, you can simply set a breakpoint in the implementing class.

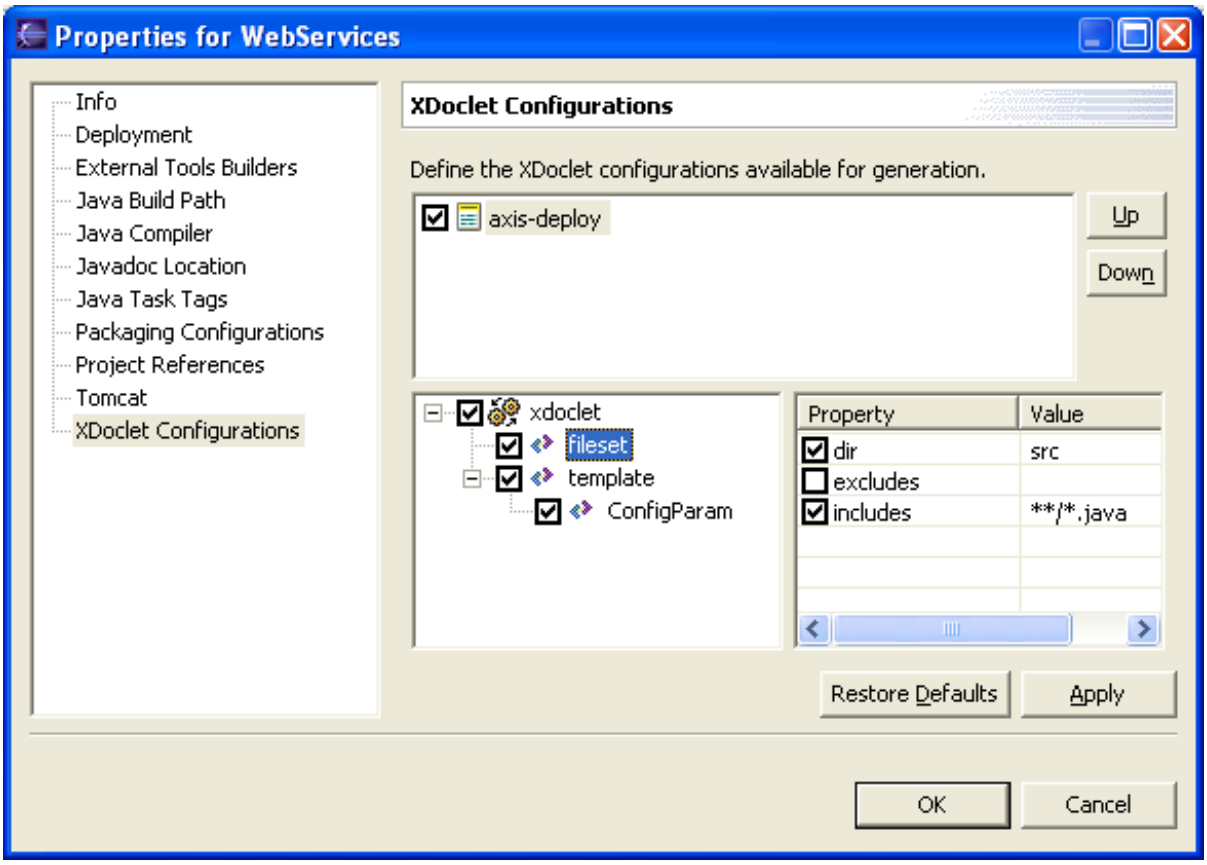
Figure 9.2. debeug a web service



9.5. XDoclet with JBOSS−IDE

The plugin JBOSS−IDE gives an alternative way to control XDoclet. Instead of using ant scripts to define and run doclets you can use a graphical user interface.

Figure 9.3. GUI for XDoclet configurations



Click, drag and assemble your XDoclet task in the section XDoclet Configurations in project properties . In the long run you will express exactly the same as described in the ant script, just with the help of this GUI. This might suit beginners better, as you do not have to think about certain details like where the XDoclet jars are. In addition all possible parameters are indicated, so you do not have to remember the exact name.

You can take a look at the generated configuration in the sample project by opening the GUI again. Alternatively throw a glance at `XDoclet-build.xml` (in the root folder of the project), that is the file where JBOSS IDE stores this configuration.

or running XDoclet with such a configuration there is even a menu entry in the context menu of the project.

9.6. New XDoclet version

While writing this article a new XDoclet version (v 1.2 not beta 2) was issued. This version also contains a new subtask axisdeploy , which can be called from within ejbdoclet. Unfortunately this new feature is not documented on the website and I found the only hint in the Manning book XDoclet in Action . This subtask generates a deployment descriptor so that this functionality overlaps with the template introduced here.

Chapter 10. New XDoclet tags

Beside the actual class name, the deployment descriptor of Axis describes a method list, a request flow etc. (see Axis documentation). To support all these attributes some new tags are necessary describing all these Axis attributes. Here the introduced template for XDoclet comes into play to process the tags.

10.1. Class level tags

10.1.1. axis.service

Declares a class as web service.

Attribute	Description	Default / Optional
name	The name of the web service	− / mandatory
scope	The scope of the web service. Possible values: request, session, application. The request scope creates a new instance of the java class for each request, application uses a singleton, which serves all requests and sessions and creates a new instance for each new client that supports sessions.	request / optional
urn	The web service name to be called. If no name is given, the class name will be used.	ClassName / optional
enable−remote−admin	If set to true, the service can be administered remotely provided that the remote admin feature is globally enabled in Axis.	false / optional
include−all	If set to true, all public methods will be exported as web service methods.	false / optional

10.1.2. axis.handler

Declares the class as request handler. The class must extend `org.apache.axis.handlers.BasicHandler`.

Attribute	Description	Default / Optional
name	The name of the handler	− / mandatory

10.1.3. axis.useHandler

Defines the handler which should be used by the web service. XDoclet then inserts a request flow element in the deployment descriptor, which links the handler to this web service.

Attribute	Description	Default / Optional
name	The name of the handler to use	− / mandatory

10.1.4. axis.parameter

Declares a parameter the handler class can request.

Attribute	Beschreibung	Default / Optional
name	Name of the parameter	

		− / mandatory
value	Value of the parameter	− / mandatory

10.2. Method level tags

10.2.1. axis.method

Declares a web service method. In this case, the include−all=true attribute at class level is not given, each method can be seperately used as web service method or not. Axis Classpath is always set to the Plugin−Libs by Lomboz, replace by the all current Libs. For 4.1.29 you need to set Tomcat 5.x in the Sysdeo−Plugin..Axis Classpath wird von Lomboz immer auf die Plugin−Libs gesetzt, ersetzen durch die kompletten aktuellen Libs. Achtung bei 4.1.29 muss man im Sysdeo−Plugin bereits Tomcat 5.x einstellen.

Chapter 11. WebService Security

In diesem Kapitel will ich kurz auf den Einsatz von Handler in Axis eingehen. Neben den in den Beispielen von Axis enthaltenen Logging−Handling, ist der Einsatz des WebService Security Standards sehr gut geeignet, um zu zeigen wie Handler in Axis funktionieren.

Ein Handler klinkt sich in die Verarbeitungsreihenfolge ein und kann den Request verändern, bevor der WebService angesprochen wird. Ebenso wird auch die Antwort wieder durch den Handler geschickt, wodurch man die Antwort ebenfalls beeinflussen kann.

Im Falle von WebService Security wird dies benutzt, um die Kommunikation mit dem WebService sicher zu machen. Man verwendet dabei nicht entwa eine Transport−Verschlüsselung wie https, die den Transport der SOAP−Nachrichten verschlüsselt, sondern man verschlüsselt die Nachricht selbst.

Dies kann in Axis mit einem Handler geschehen, der sich in den Request−Ablauf einklickt. Dabei wird die Anfrage vor dem Verschicken verschlüsselt und serverseitig vor der Verarbeitung wieder entschlüsselt. Genau umgekehrt wird die Antwort serverseitig wieder verschlüsselt und dann auf dem Client wieder entschlüsselt.

11.1. Handler für WebService Security

Der Artikel Axis sicher aus dem Java−Magazin diente als Vorlage, um diesen Handler im Beispiel−Projekt zu bauen. Ich habe nur einige wenige Veränderungen vorgenommen, um das Projekt leichter in Betrieb nehmen zu können[5].

11.1.1. Zusätzliche Bibliotheken

Zum Einsatz kommen hier die im Artikel des Java−Magazins vorgeschlagenen Bibliotheken von IBM und VeriSign. IBMs Security Bibliothek ist sowohl in IBMs WebSphere SDK for Web Services (WSDK) als auch im Emerging Technologies Toolkit (ETTK) enthalten. Von den beiden recht grossen Downloads wird allerdings nur ibmjceprovider.jar und die ibmjlog.jar benötigt. Ich habe mir deshalb die Freiheit genommen und diese Bibliotheken direkt ins Beispiel−Projekt aufgenommen. Der reguläre Download umfasst viele Megabyte mehr an Daten und erfordert auch das Akzeptieren der Lizenzbedingungen näheres siehe:

Download WSDK from IBM:

http://www−106.ibm.com/developerworks/webservices/wsdk/

alternativ Download ETTK:

http://www.alphaworks.ibm.com/tech/ettk

Entsprechendes gilt für die Bibliotheken von VeriSign. Auch hier werden einfach nur die beiden Bibliotheken benutzt. Man kann die kompletten Pakete downloaden und nach der Installation die Bibliotheken kopieren und dann alles wieder entfernen.

Downloads von VeriSign:

VeriSign Trust Services Integration Kit (TSIK): http://www.xmltrustcenter.org/developer/verisign/tsik/index.htm

und

WS−Security Implementierung von VeriSign:
http://www.xmltrustcenter.org/developer/verisign/dist/ws−security−1.7.zip

11.1.2. Vergleich der SOAP−Nachrichten

Schaut man sich die SOAP Nachrichten an, die bei der gesicherten Kommunikation ausgetauscht werden, so fällt auf dass die gesicherte Variante erheblich grösser ist und somit einigen Overhead erzeugt.

Vorher (ungesichert):

```
<soapenv:Envelope

xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance">

<soapenv:Body>

<ns1:getAddress

soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

xmlns:ns1="http://service.solutions.rinke.de">

<id xsi:type="xsd:int">0</id>

</ns1:getAddress>

</soapenv:Body>

</soapenv:Envelope>
```

Nachher (gesichert):

```
<soapenv:Envelope

xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance">

<soapenv:Header>

<wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext">

<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">

<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa−1_5"/>

<xenc:CipherData>

<xenc:CipherValue>
```

```
            dMjLSkUIzeDo7MxKg6lXtgQOwW2ZFDUUEhF/cKNY475FaxsCSwtAgfpMBlTBHf+jFYxIIq95
            29oVLNG8Sq8mbwYOCbEdkpIi7Eb8P8jV+rYn/Qiy12C6an0pqo5U18tffv2mDvnYW1V+SDVV
LiKR7ih2cGeY+AT0IiR0np1Apho=
</xenc:CipherValue>
</xenc:CipherData>
<xenc:ReferenceList>
<xenc:DataReference URI="#wsse-c6cf62d0-5ef8-11d8-8ffa-ada096a18c18"/>
</xenc:ReferenceList>
</xenc:EncryptedKey>
<wsse:BinarySecurityToken ValueType="wsse:X509v3"
EncodingType="wsse:Base64Binary"
wsu:Id="wsse-c64f0f40-5ef8-11d8-8ffa-ada096a18c18"
xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
            MIIB2TCCAUICBEAtZtMwDQYJKoZIhvcNAQEEBQAwNDELMAkGA1UEBhMCREUxFDASBgNVBAoT
            C1N0ZWZhblJpbmtlMQ8wDQYDVQQDEwZDbGllbnQwHhcNMDQwMjE0MDAwNzQ3WhcNMDQwNTE0
            MDAwNzQ3WjA0MQswCQYDVQQGEwJERTEUMBIGA1UEChMLU3RlZmFuUmlua2UxDzANBgNVBAMT
            BkNsaWVudDCBnjANBgkqhkiG9w0BAQEFAAOBjAAwgYgCgYB24AKhC8OqeTi0eXn2EP6uKWBe
            sIvye7q0wnewZ/X0uGXbBRvX0PH8JkTyopqwjwVMnJaGnbFbuQEezxLo5kctdSNoQSB628m+
            LT2rDkhiZqg5jIWqKIQXqXgg8EuBAg9hgoaR2Y3xex7eri72CfC7u2ICaPkqmIBlfkVZIwkK
            FwIDAQABMA0GCSqGSIb3DQEBBAUAA4GBAAgDbm4+zbK50QGACEHO2v+bM/v2uN/3wq5DxqNL
            wI1bgsRSVMGCaAq6HZUCTdqBDCcJlk2dOG+PETM0tP+bh/jM3yIpccV75qs3NTOQHZIP5v9Z
            ZbIb3wcHBLcJCuxf6ZLe/b6IuibcQfz4JwuxifBhG0BbKMTuCvk7adqcPawx
</wsse:BinarySecurityToken>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<ds:Reference URI="#wsse-c60730d0-5ef8-11d8-8ffa-ada096a18c18">
<ds:Transforms>
```

```
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>VO6yW5RrFZA+sr3QYDFYjIcyuwI=</ds:DigestValue>
</ds:Reference>
<ds:Reference URI="#wsse-c5ff8fb0-5ef8-11d8-8ffa-ada096a18c18">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>CYvLe0Wv0mJ6bPvGfapUP5ahtgU=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>
           VvClhOxjXyIlrGi3BS9yc4Nm/40N+/vKgHIxtjTOVIueqggVhSym7U01C6RC1je+WS1yRNdU
           9eGoKTlPrNbqgL4txN3X5OTDhJaxMpf6IGdsVd+UikXJoN0+OcmJVRQd4bUqraIZrmeEWbVz
ZGACIe297cBZZ1KTHary6z1Vs9Y=
</ds:SignatureValue>
<ds:KeyInfo>
<wsse:SecurityTokenReference>
<wsse:Reference URI="#wsse-c64f0f40-5ef8-11d8-8ffa-ada096a18c18"/>
</wsse:SecurityTokenReference>
</ds:KeyInfo>
</ds:Signature>
</wsse:Security>
<wsu:Timestamp xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
<wsu:Created wsu:Id="wsse-c5ff8fb0-5ef8-11d8-8ffa-ada096a18c18">
         2004-02-14T14:19:17Z
</wsu:Created>
```

```
</wsu:Timestamp>

</soapenv:Header>

<soapenv:Body wsu:Id="wsse-c60730d0-5ef8-11d8-8ffa-ada096a18c18"

xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">

<xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Content"

xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"

Id="wsse-c6cf62d0-5ef8-11d8-8ffa-ada096a18c18">

<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>

<xenc:CipherData>

<xenc:CipherValue>

           uMNGMbCMiAruKyx42I1C/5iYQxnt1W575l/Dvpm2v2cRIN0JgQtCPQQxmhYhXePH28dtct+W

           V2VzWeNDrd2LjGjh5tiRMOecZLQ78rQ7/uy6aLn/OcLOiOucJ61kB2+B1V36P7R1a64Gizak

           u54FB4qQJloWrme5FFqdVVh3c2PS0gtKcefoYIrPygmhtbayFRJNyZiRxvSDrrlYuTYgsTmN

d3hQuWKTr9cOPXExWp31NJ8Mp/Ir2/XgNawrPVTurF2qTOa+tsw=

</xenc:CipherValue>

</xenc:CipherData>

</xenc:EncryptedData>

</soapenv:Body>

</soapenv:Envelope>
```

Wenn man diese enorm grosse XML−Message anschaut, fragt man sich allerdings, ob nicht besser doch ein binäres Protokoll angezeigt wäre ;−). Einer der Vorteile − nämlich die Lesbarkeit der XML−Nachrichten ist hier jedenfalls nicht mehr vorhanden.

---

[5] Es wurden einige absoluten Pfadangaben entfernt und die Security−Provider werden dynamisch registriert.

Chapter 12. Download WebServices withJava, Axis, XDoclet and Eclipse

- Sample project
- PDF
- HTML
- Microsoft Word
- Windows Help
- Plaintext
- HTML (one file)
- XML (DocBook)
- OpenOffice
- Eclipse−Plugin
- Build Framework (to write such articles with DocBook)

Chapter 13. Appendix

13.1. Online references

13.1.1. Eclipse plugin directories

- http://www.crionics.com/products/opensource/eclipse/eclipse.html
- http://eclipse−plugins.2y.net/eclipse/index.jsp

13.1.2. More online articles

- http://www.oio.de/public/warum−eclipse.htm
- http://www.3plus4software.de/eclipse/index.html
- http://www.capescience.com/articles/index.shtml

13.1.3. General online resources for Eclipse

- http://www.eclipse.org/
- http://www.eclipseproject.de/

13.1.4. More plugins for web services with Eclipse

- WASP Developer from Systinet
- WebService SDK from IBM
- WebService Pack from Sun