Inhaltsverzeichnis

Inhaltsverzeichnis

WebServices mit Java, Axis, XDoclet und Eclipse

Version 1.00

26. Februar 2004

Diesen Artikel gibt's online auf Stefan Rinke, Articles. Dieser Artikel wurde mit OpenOffice erstellt und anschließend nach DocBook konvertiert. Schließlich wurden mit dem DocBook−Framework die unterschiedlichen Ausgabeformate (siehe Kapitel 12, Download) erzeugt. Das DocBook−Framework selbst ist im Artikel: DocBook−Publishing beschrieben.

Kapitel 1. Einführung

Wer heute in einer J2EE Umgebung WebApplications und insbesondere WebServices entwickeln möchte, kann auf sehr leistungsfähige Tools zurückgreifen, die noch nicht einmal Lizenzgebühren kosten.

Mit Eclipse als IDE, Tomcat oder JBOSS als J2EE−Application−Server, Axis als WebServices−Framework und XDoclet hat man mächtige Tools zusammen, die ein effizientes Entwickeln von WebServices möglich machen.

Dieser Artikel soll zeigen, wie eine komplette Umgebung aufgebaut ist und welche Plugins für Eclipse benötigt werden, damit man schnell und reibungslos arbeiten und sich auf die eigentliche Geschäftlogik konzentrieren kann. Es werden alle Plugins vorgestellt und ein Beispielprojekt entwickelt, welches einen WebService mit Axis und Tomcat realisiert. Es wird ein XDoclet−Template vorgestellt für die Erzeugung des Deployment−Descriptors und ein Ant−Script zum Deployment. Schließlich sieht man wie man in Eclipse einen WebService debuggen kann.

Verwendet werden hier: Eclipse 2.1.1, Tomcat 4.1.29, Sysdeo Tomcat Plugin 2.1.0, Lomboz−Plugin 2.1.2, Axis 1.1 und XDoclet 1.2 b3.

Für WebService Security: VeriSign Trust Gateway 1.1 vom 26.09.2003, ws−security 1.7. Von IBM (für die zusätzlichen Security−Provider: IBM Emerging Technologies Toolkit Web Services 1.2. Die Tools von VeriSign und IBM müssen nicht heruntergeladen werden, die die betreffenden Bibliotheken bereits im Beispiel−Projekt enthalten ist.

Eine weitergehende Einführung in Java, J2EE, Tomcat oder JBOSS ist nicht Gegenstand der folgenden Ausführungen. Hier sei im Anhang auf weiterführende Quellen verwiesen.

Kapitel 2. Eclipse: eine modulare Entwicklungsumgebung

Mit Eclipse hat IBM ein echtes Meisterstück abgeliefert. Durch das modulare Design lässt sie sich beliebig erweitern und somit auf unterschiedlichste Bedürfnisse anpassen. Als OpenSource−Produkt erfreut sie sich großer Beliebtheit, was sich vor allem in der großen Zahl an Erweiterungen niederschlägt. So findet man für alle erdenklichen Sprachen und typischen Entwickleraufgaben Plugins [1], die einem das Leben erleichtern.

Da Eclipse selbst in Java geschrieben ist, spielt es vor allem im Javaumfeld seine Stärken voll aus. Hier sind auch die meisten Plugins verfügbar.

2.1. Installation

Eclipse lässt sich einfach aus dem Archiv auspacken und unter einem beliebigen Pfad installieren (unter Windows z.B. `C:\Programme\Eclipse`) unter Linux (Suse 8.1) unter `/opt/eclipse` . Wichtig für Suse 8.1: die Bibliothek gtk 2.0 muss installiert sein.

Alle Plugins liegen unter dem Installationsverzeichnis im Verzeichnis `plugins`, darunter die einzelnen Packages mit umgekehrten Domainnamen als Verzeichnisse (z.B. `org.eclipse.jdt&`).

Um weitere Plugins zu installieren genügt es, diese entsprechend in die Verzeichnishierarchie zu entpacken und dann Eclipse neu zu starten. Neue Plugins werden dann automatisch erkannt und eingerichtet.

Weiterführende Hinweise zu Eclipse findet man entweder unter Java WebApplications mit Eclipse .

---

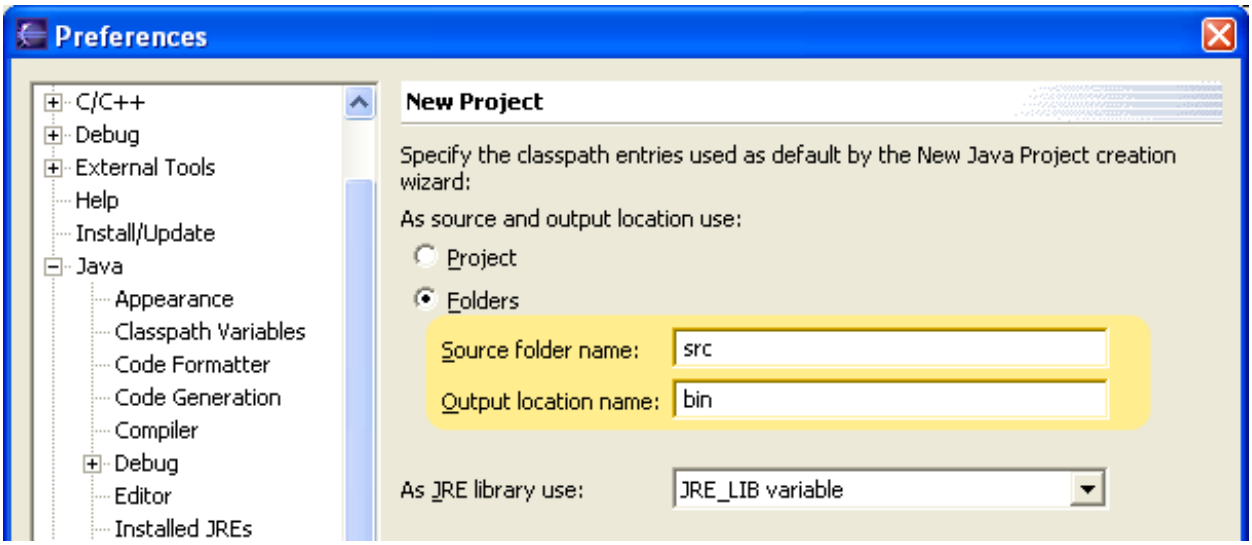
[1] Ein Verzeichnis der vielen Plugins findet man z.B. unter http://eclipse−plugins.2y.net/eclipse/index.jsp oder siehe Abschnitt 13.1.1

Kapitel 3. Javaprojekte mit Eclipse

3.1. Grundeinstellungen

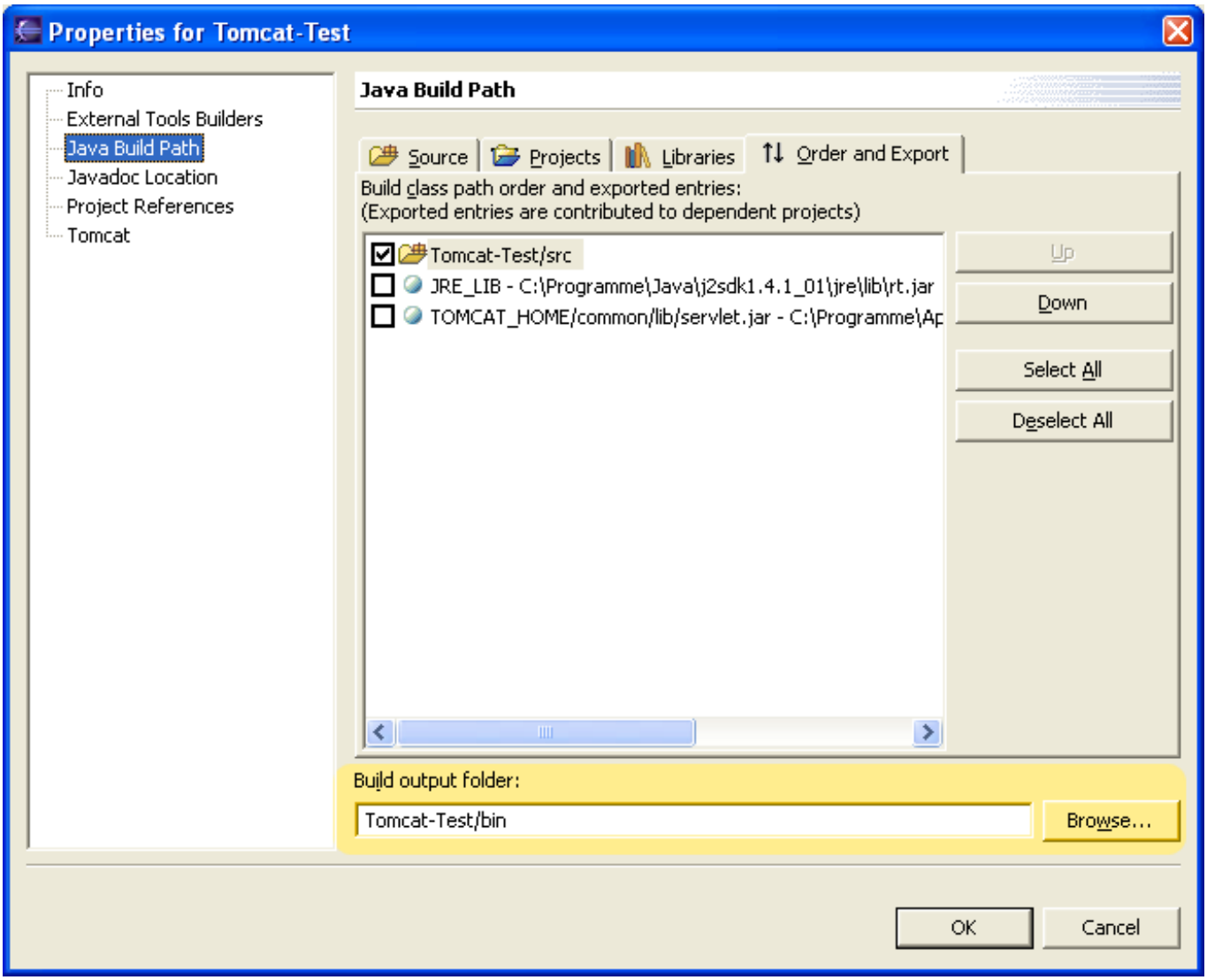
Zunächst einmal sollte man dafür sorgen, dass die Sourcecodes grundsätzlich von den erzeugten Class−Files getrennt bleiben. Hierfür ändert man im Dialog Windows − Preferences den Source−Folder−Name auf src und die Output−Location auf bin .

Abbildung 3.1. Preference New Project



Für WebServices ist es evtl. sogar besser das Output−Verzeichnis gleich so einzustellen, dass alles Class−Files dort landen, wo der Applicationserver sie erwartet (`WEB-INF/classes`). Das kann man aber auch in der Project−Preferences für jedes Projekt getrennt einstellen.

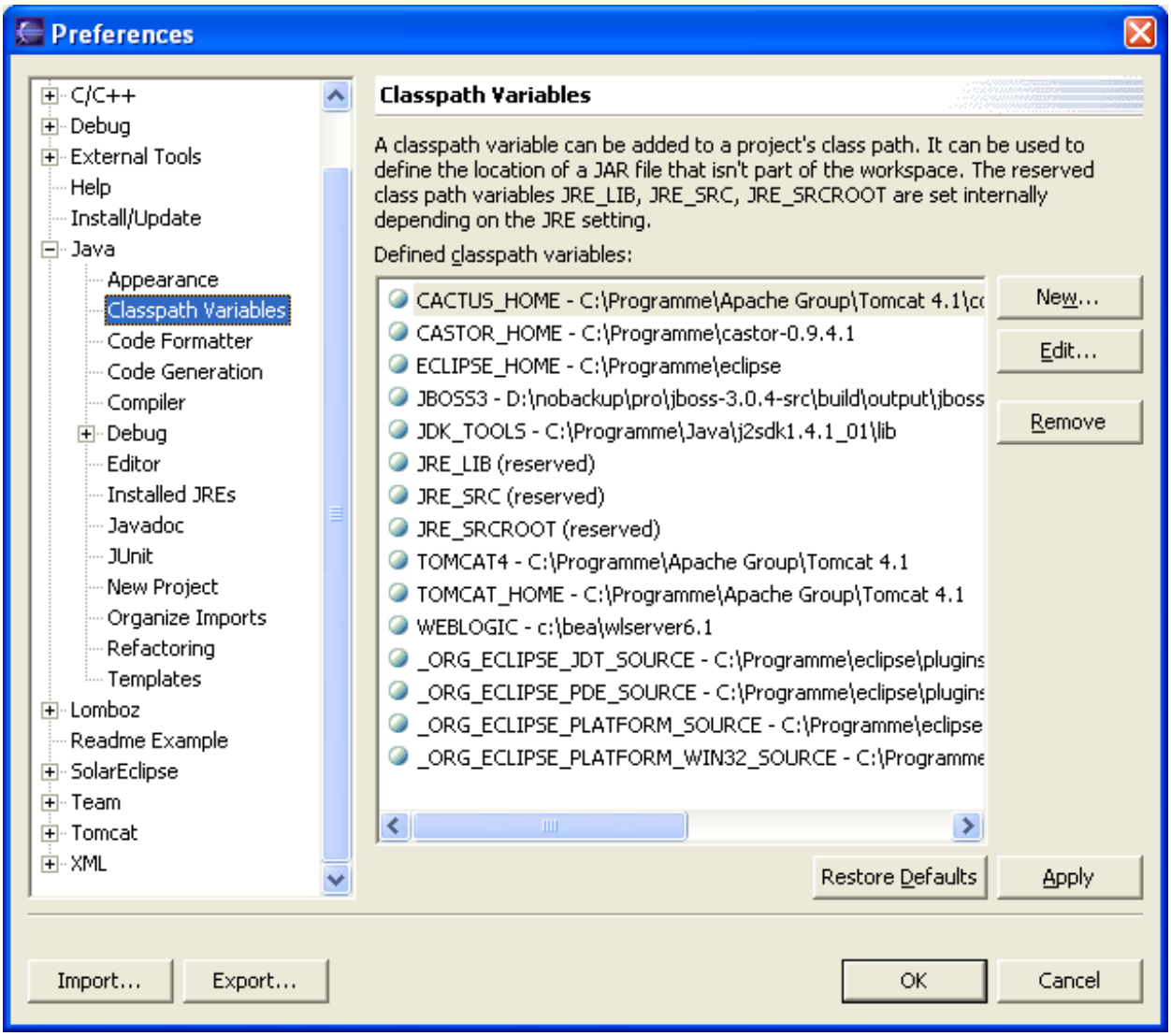
Abbildung 3.2. Properties Build output folder



3.1.1. Classpath−Variablen

Außer der Laufzeit−Bibliothek muss man bei praktisch jedem Projekt andere Bibliotheken mit einbinden. Da diese bei jedem Entwickler auf unterschiedlichen Pfaden installiert sein können, benutzt man sogenannte Classpath−Variablen.

Abbildung 3.3. Preferences Classpath Variables



Im Projekt werden so nur symbolische Namen gespeichert, die jeder Entwickler auf seine lokale Installation anpassen kann. Braucht man z.B. servlet.jar von Tomcat, dann wird sie im Projekt als TOMCAT_HOME/common/lib/servlet.jar referenziert. Wo Tomcat dann genau installiert ist, wird vom Entwickler selbst festgelegt in dem er den Wert von TOMCAT_HOME entsprechend seiner lokalen Installation einstellt.

3.1.2. Versionierte Namen

Wenn man in einem Team entwickelt und relativ neue Technologien im Einsatz hat, dann ändern sich Versionen der zugehörigen Bibliotheken sehr schnell. Damit nicht das heillose Chaos ausbricht und seltsame Effekte auftreten, die auch noch von einem zum anderen Teammitglied unterschiedlich ausfallen, ist es sehr wichtig, dass alle immer die gleichen Versionen alle Komponenten verwenden.

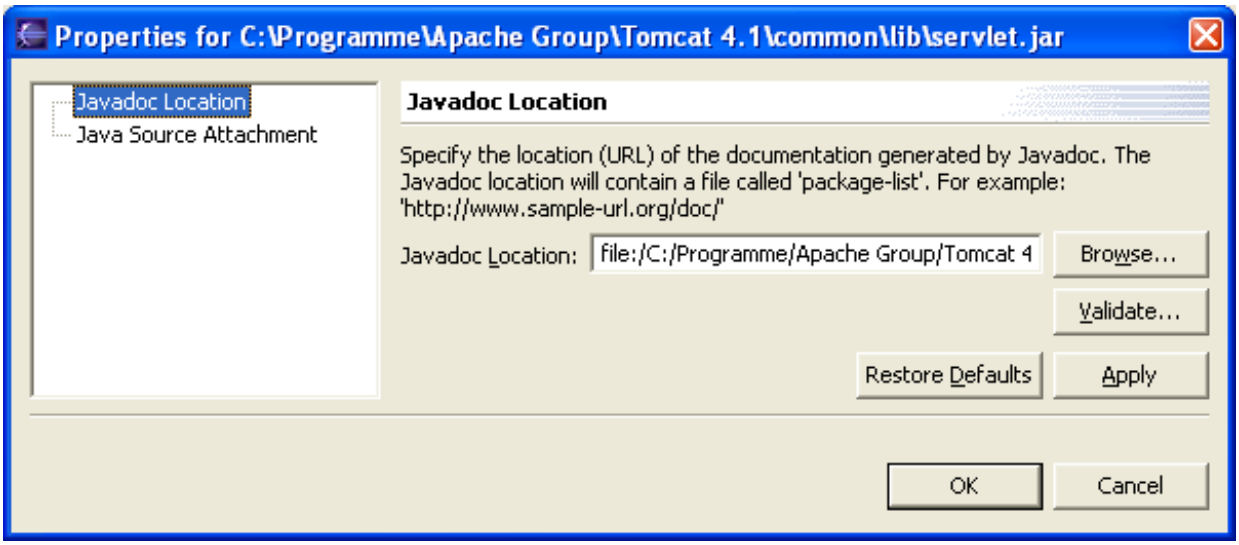
Deshalb empfehle ich immer Namen für Bibliotheken zu verwenden, die schon im Namen die Versionsnummer enthalten. z.B. castor−0.4.1.jar . Wenn alle Bibliotheken auch so im Projekt referenziert werden, ist sichergestellt, dass alle das Gleiche verwenden.

3.1.3. Erweitern der Dokumentation

Man kann die eingebaute Dokumentation sehr leicht erweitern mit eigenen Texten erweitern. Zum einen lassen sich alle mit JavaDoc erzeugten API−Dokumentationen direkt einbinden. Das gilt zu allererst natürlich für das Java−SDK selbst, aber dann natürlich auch für alle im Projekt benutzten Bibliotheken.

Zum anderen können Plugins ihre Dokumentation nahtlos integrieren, als Beispiel gibt's im Download−Bereich diesen Artikel auf als Plugin, wo durch er eben in der Eclipse Hilfe erscheint.

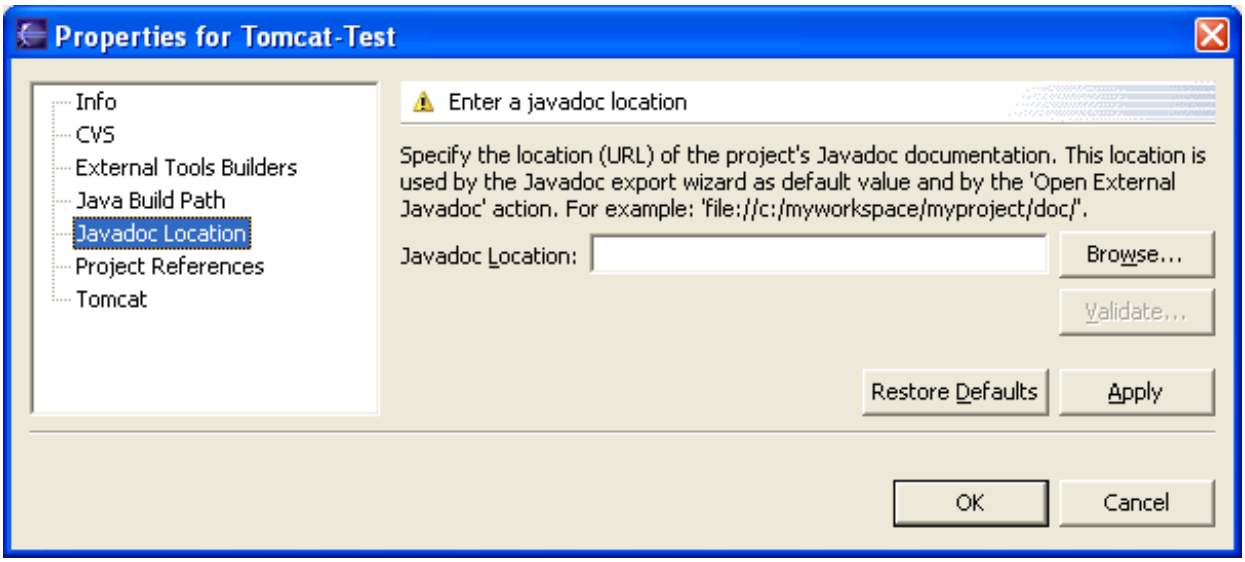
Abbildung 3.4. Properties für eine Bibliothek



Um die Möglichkeiten voll auszunutzen, sollte man zu jeder verwendeten Bibliothek (JAR−File) die entsprechende API−Dokumentation, sowie den Sourcecode (soweit verfügbar) zuordnen. Wenn man ein komplettes JDK installiert inklusive Source erkennt Eclipse das automatisch. Für alle individuellen Bibliotheken ordnet man über die Properties eines jeden JARs (erreichbar über das Kontextmenü) die Dokumentation und das Source−Archiv zu.

Entsprechend wird auch die API−Dokumentation des eigenen Projekts mit eingebunden:

Abbildung 3.5. Properties eigene Dokumentation einbinden



So vorbereitet liefert den Druck auf Shift−F2 jederzeit die Dokumentation zum Objekt unter dem Cursor, genauso wie F3 das betreffende Sourcefile öffnet.

Kapitel 4. WebServices

Mit den WebServices gibt es nunmehr nach CORBA, DCOM, EJB eine weitere Möglichkeit distributed Computing also verteilte Anwendungen zu realisieren. WebServices definieren einen weiteren Standard wie unterschiedliche Softwarekomponenten über das Netzwerk zusammenarbeiten können. Dabei ist besonderes Augenmerk auf die Plattformunabhängigkeit gelegt worden, d.h. WebServices sollen möglichst hersteller− und systemunabhängig mit einander arbeiten können. Als Verbindungsnetzwerk ist von Anfang an das Internet eingeplant gewesen, das Ziel soll also sein Anwendungen mit WebServices über das Internet zusammen zu schalten.

Ein weiterer wichtiger Punkt ist: Microsofts .NET Framework verwendet WebServices als integralen Bestandteil seiner verteilten Infrastruktur. Es wird also nicht mehr DCOM verwendet sondern in .NET gibt es WebServices an allen Ecken und Enden.

4.1. WebServices Nachrichten

WebServices Nachrichten verwenden eigentlich immer den SOAP−Standard, welcher wiederum eine XML−Anwendung darstellt. Anfragen an einen WebService werden also in XML ausgedrückt und auch die Antworten, die ein WebService zurückliefert sind mit XML beschrieben. Ich werde später zeigen, wie man sich bei der Fehlersuche mit Axis solche SOAP−Messages anschaut.

4.2. WebServices Transport

Der Transport von SOAP−Messages für WebServices geschieht fast immer über HTTP(S). Wie eingangs erwähnt ist das Internet das bevorzugte Transportnetzwerk für WebServices, weshalb HTTP als Transport natürlich sehr gut passt. Verwendet man HTTP kann man in der Regel auf eine sehr gute Infrastruktur zurückgreifen, da HTTP an anderer Stelle eben für die WWW−Dienste fast immer schon eingesetzt wird. Dementsprechend hat man weniger Probleme mit Firewalls und dergleichen: HTTP geht immer! .

4.3. WSDL WebService Definition Language

Mit WSDL − einem W3C−Standard − wird ein WebService beschrieben. Die Beschreibung ist auch in XML formuliert und definiert ähnlich wie z.B. die Interface Definition Language IDL der OMG das Interface des WebServices. Das WSDL−Dokument beschreibt also im Wesentlichen welche Methoden der WebService anbietet, mit welchen Parametern sie aufzurufen sind und was sie zurückliefern.

4.4. Axis

Das Axis−Framework von Apache (http://xml.apache.de/axis) liefert die notwendige Basis zum Erstellen von WebServices in Java. Mit Axis kann man zum einen sehr einfach WebServices benutzen, d.h. als Client einen WebService aufrufen. Zum Anderen stellt Axis aber auch sehr viel Unterstützung bereit, wenn man selbst WebServices schreiben möchte.

Axis kann den WebService nicht selbst publizieren, sondern läuft als Servlet in einem Servlet−Container wie Tomcat oder JBOSS.

Kapitel 5. XDoclet

Wie bei fast allen J2EE Technologien hat man auch bei WebServices das Problem, dass neben den eigentlichen Java−Klassen immer auch eine Beschreibung (meist in XML) formuliert werden muss, die neben der reinen Implementierung Attribute dessen beschreibt, was da in Java programmiert wurde. Man spricht von sogenannten Deployment−Descriptoren die bei der Inbetriebnahme (Deployment) gebraucht werden, um bestimmte Eigenschaften festzulegen. Dies ist bei Enterprise JavaBeans so und eben auch bei WebServices[2].

Um die Schwierigkeit zu vermeiden immer mehrere Dateien ändern zu müssen, wenn sich z.B. ein Klassennamen ändert, geht man bei der Attribut−Orientierten−Programmierung mit XDoclet davon aus, dass alle Informationen inklusive aller Attribute direkt im Java−Quellcode enthalten sind. Die Metadateien werden dann bei einer Änderung einfach von XDoclet neu erzeugt. Auf diese Weise sind immer alle Dateien konsistent und man behält auch besser den Überblick, weil man nicht an N Stellen nachlesen muss, was eine Klasse genau macht.

5.1. XDoclet Kommentare

XDoclet benutzt JavaDoc Kommentare, um die zusätzlichen Attribute im Java−Quellcode unterzubringen. Beispiel:

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

(1) Hier wird mit einem XDoclet Kommentar ein WebService deklariert.

5.2. XDoclet Templates

Zum Erzeugen der Metadateien wie Deployment−Descriptoren werden Templates benutzt. XDoclet definiert hier seine eigene Template−Sprache, die genau auf die Anforderungen zugeschnitten ist. Dieses Beispiel−Template gibt für alle Klassen, die als Axis Handler deklariert sind, den kompletten Klassennamen aus:

```
<?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) Iteriere über alle Klassen, die in der Eingabemenge sind (das angegebene Fileset).

(2) Wenn eine Klasse ein Class−Level−Tag mit Namen axis.handler und einem Parameter name hat, dann ...

(3) gib von dieser Klasse den kompletten Namen aus.

Alle Tags die mit XDt beginnen, gehören zur Template−Sprache von XDoclet und sind in der XDoclet−Dokumentation erklärt.

[2] Die Liste läßt beliebig verlängern: Servlets, Taglibs, OR−Mappings, ...

Kapitel 6. Eclipse und J2EE

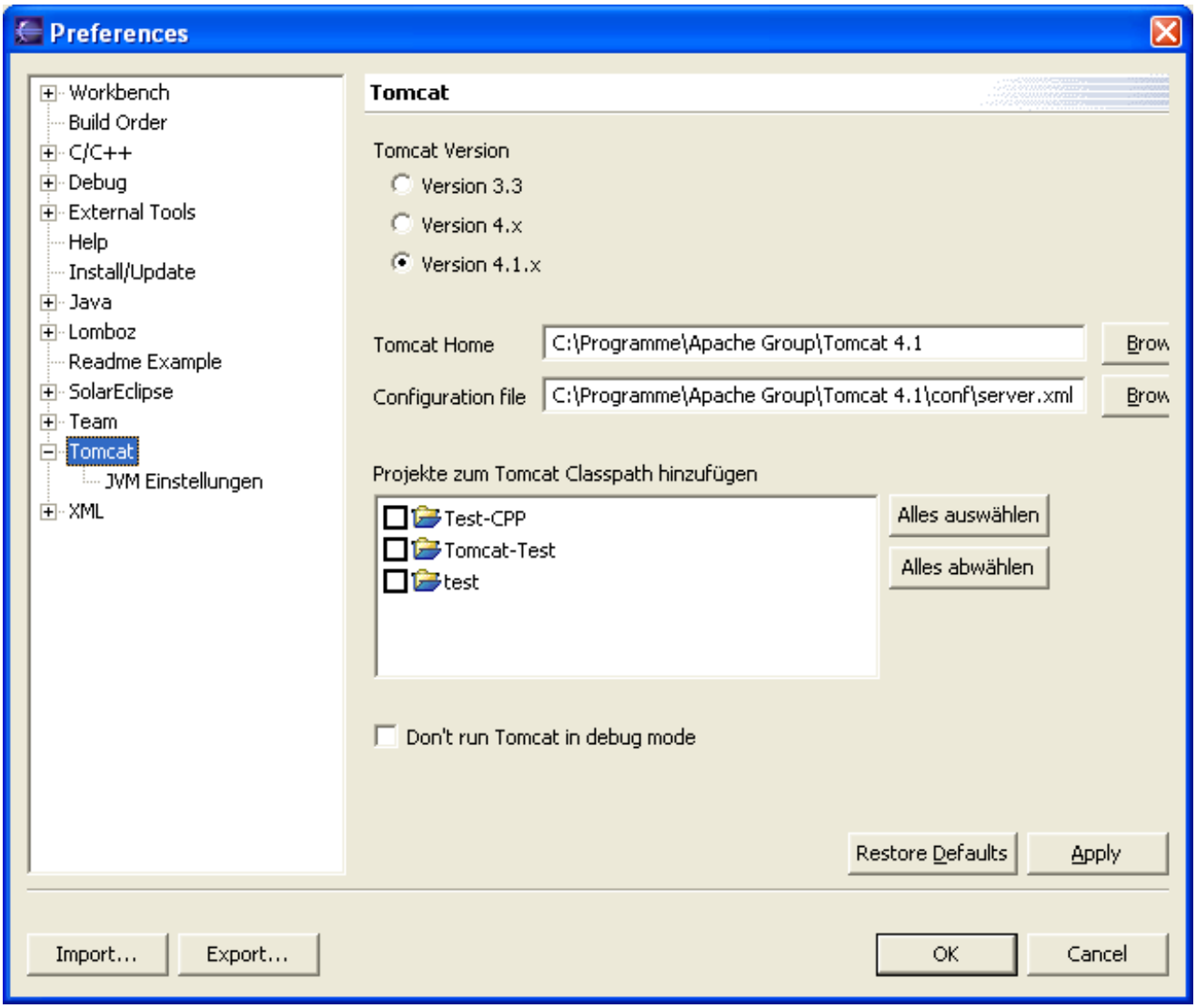
Um WebServices mit Eclipse zu entwickeln sind mehrere Plugins empfehlenswert. Das Tomcat−Plugin von Sysdeo ist für einfache Projekte mit Tomcat als Applicationserver gut geeignet. Da ich später mit Tomcat als Container WebServices entwickeln und debuggen möchte, wird es in diesem Projekt verwendet. Wesentlich mehr Funktionalitäten liefert aber das Lomboz−Plugin von ObjectLearn. Es ist primär hilfreich beim Entwickeln von EJB−Applicationen, hat aber auch ein paar Funktionen für WebServices.

Schließlich kann optional noch das Plugin JBOSS−IDE verwendet werden, um XDoclet mit etwas mehr Komfort zu steuern. Ich werde beide Arten vorstellen zu Fuss per Ant−Script und mit JBOSS−IDE.

6.1. Tomcat−Plugin von Sysdeo

Zunächst wird das ZIP−Archiv in die Verzeichnishierarchie von Eclipse entpackt. Nach dem Neustart von Eclipse läßt sich das Plugin unter Windows / Preferences konfigurieren:

Abbildung 6.1. Tomcat−Plugin Konfigurieren

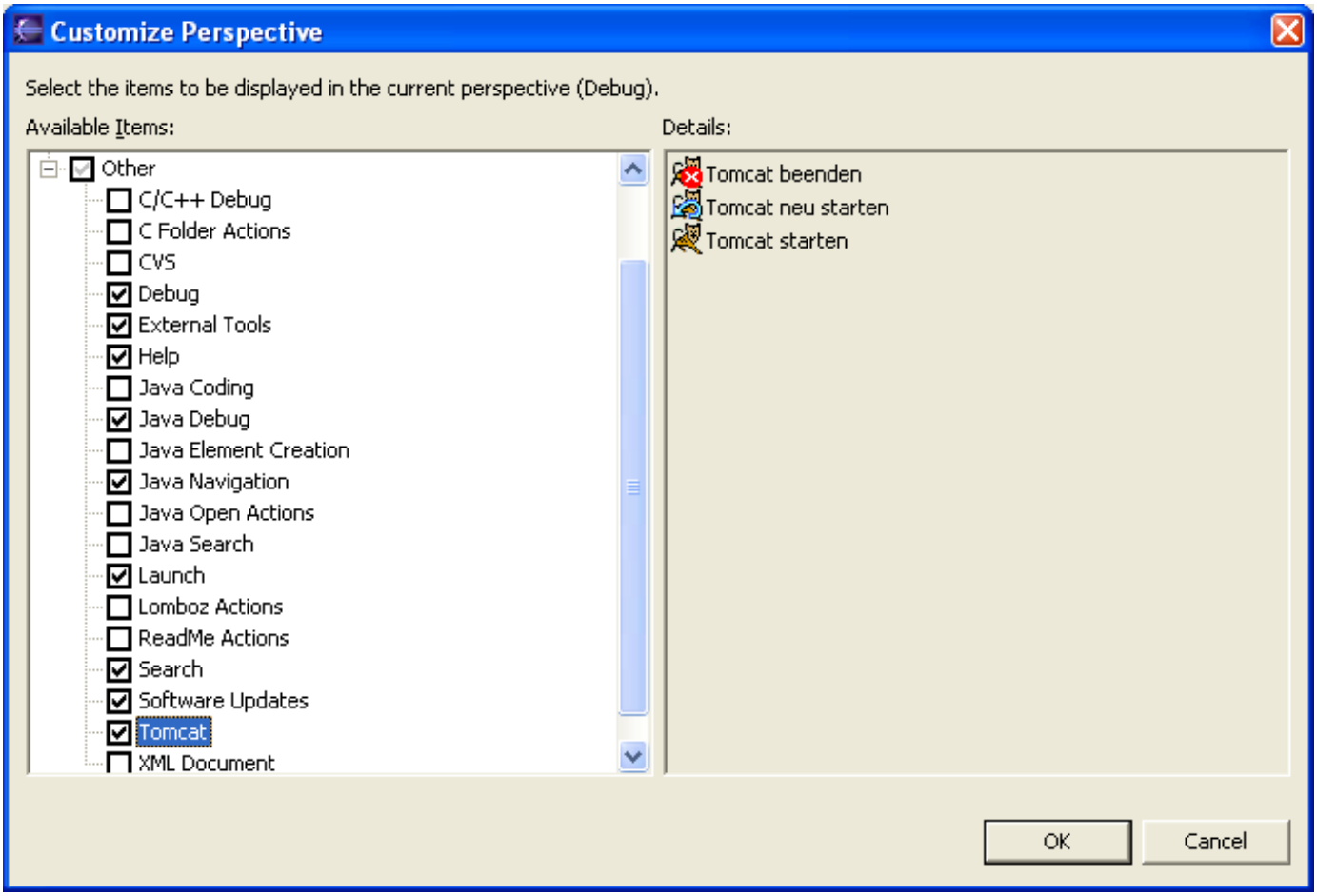


Die wichtigsten Einstellungen sind:

- Tomcat−Version
- Tomcat−Home−Verzeichnis
- Tomcat−Konfigurationsdatei

Damit kann das Plugin bereits benutzt werden. Unter Windows / Customize Perspective aktiviert man nun in der Debug−Perspective das Tomcat−Plugin:

Abbildung 6.2. Tomcat−Plugin aktivieren



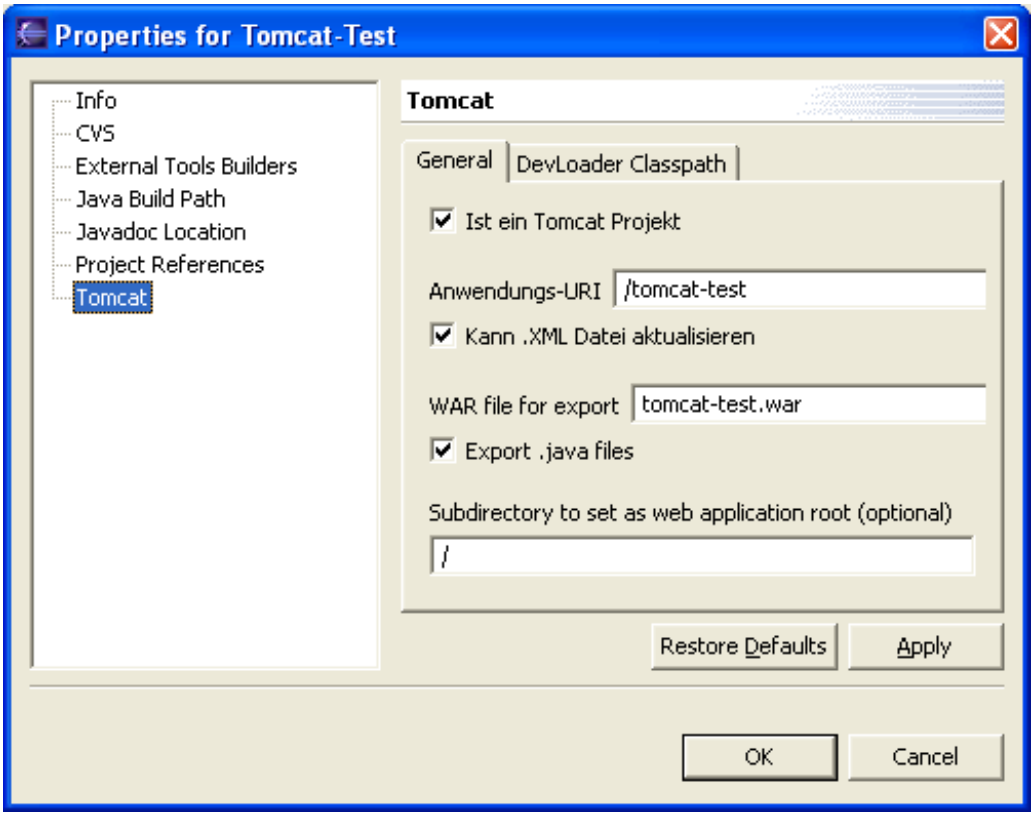
Nun erscheinen in der Toolbar drei neue Buttons  über die man Tomcat direkt unter Eclipse starten und stoppen kann.

Das interessante daran ist, dass man so eine komplette WebApplication inklusive Tomcat debuggen kann. Man kann also ohne weiteres in die Methoden des TestServlets einen Breakpoint setzen und wenn man nun mit einem Browser das Servlet aufruft, bleibt der komplette Request genau in der Servlet−Methode stehen.

6.1.1. Projekt als Tomcat−Projekt

Wenn man ein Project neu anlegt ist fortan Tomcat−Projekt neben Java−Projekt als neuer Projekttyp verwendbar. Ein bestehendes Projekt läßt unter Projekt−Properties zum Tomcat−Projekt machen:

Abbildung 6.3. Tomcat−Projekt Einstellungen

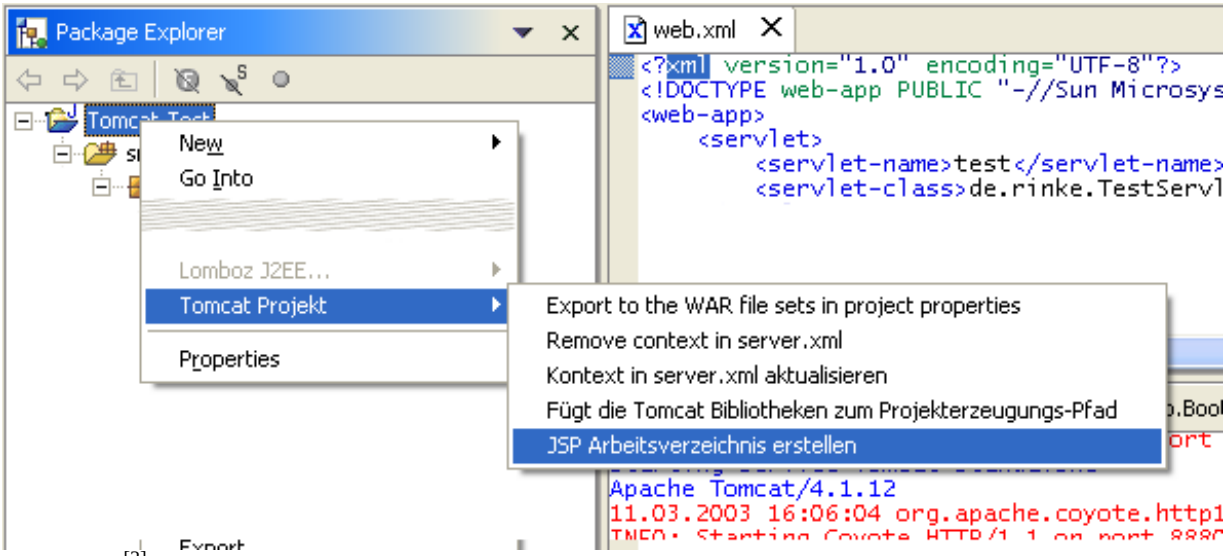


- Zunächst muss Ist ein Tomcat Project angekreuzt sein.

- Bei Anwendungs URI wird der Context−Pfad eingetragen mit dem der Tomcat−Context an gelegt werden soll.
- Wenn Kann .XML aktualisieren angekreuzt ist, dann wird der Tomcat−Context automatisch in die server.xml eingetragen.
- WAR file for export legt den Namen des WAR−Files fest, wenn das Projekt exportiert wird. Der Name ist allerdings ein absoluter Pfadnamen, d.h. der Pfad verweist direkt auf das Verzeichnis in das deployed werden soll.

Wenn alles so vorbereitet ist, dann ist im Kontextmenü des Projekts ein zusätzlicher Eintrag vorhanden:

- **Abbildung 6.4. Tomcat−Projekt Optionen im Kontextmenü**



Das WAR[3]−File läßt sich ausliefern.

- Der Context in der Tomcat−Konfigurationsdatei kann entfernt bzw. aktualisiert werden.
- Die Tomcat−Bibliotheken können auf Knopfdruck alle zum Projekt hinzugefügt werden.
- Es wird ein Arbeitsverzeichnis erzeugt, welches die aus JSP−Seiten erzeugten Servlets aufnimmt (wozu das gut ist, sieht man später unter).

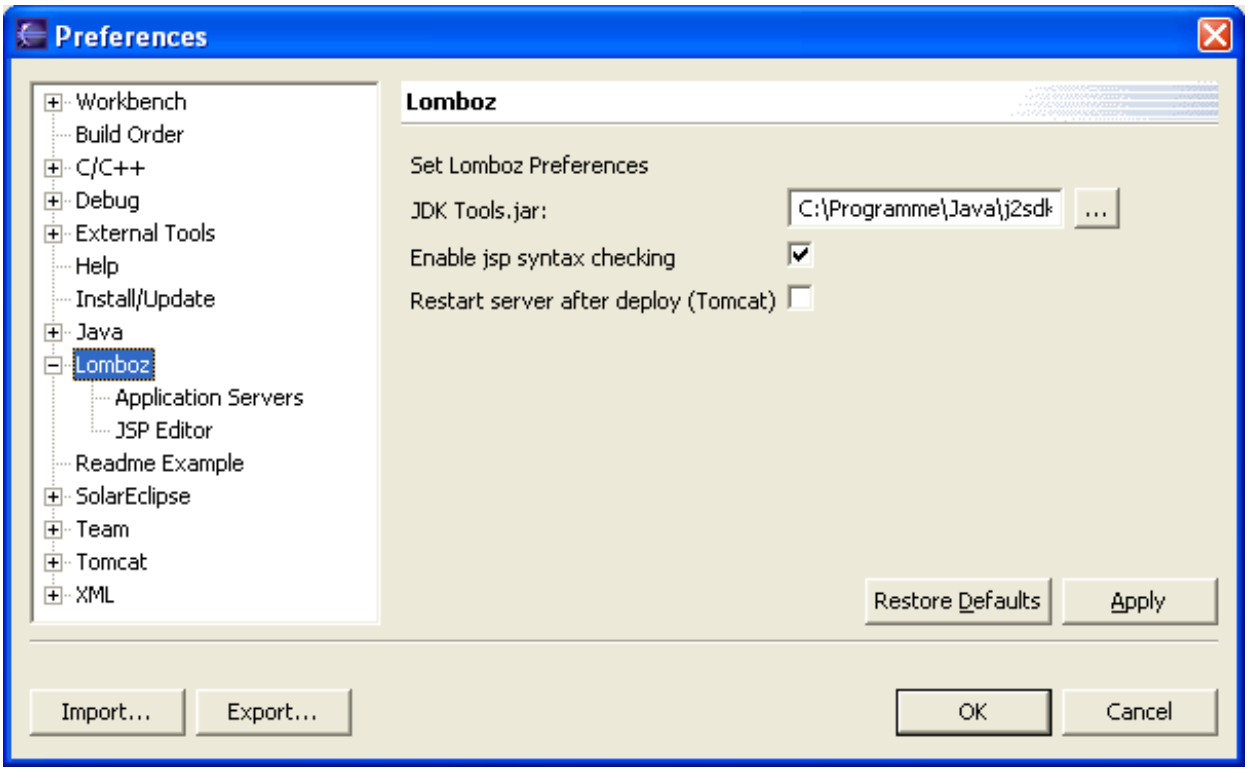
6.2. Lomboz−Plugin von ObjectLearn

ObjectLearn hat mit seinem Plugin ein sehr mächtiges Werkzeug zum Erstellen von J2EE Applicationen vorgelegt. Was hier vorgestellt wird ist nur ein kleiner Ausschnitt aus der gesamten Funktionalität, der sich mit WebServices beschäftigt. Der ganze EJB−spezifische Teil und ebenso das meiste was sich mit Servlets beschäftigt wird hier zunächst ausgespart.

6.2.1. Installieren und Einrichten

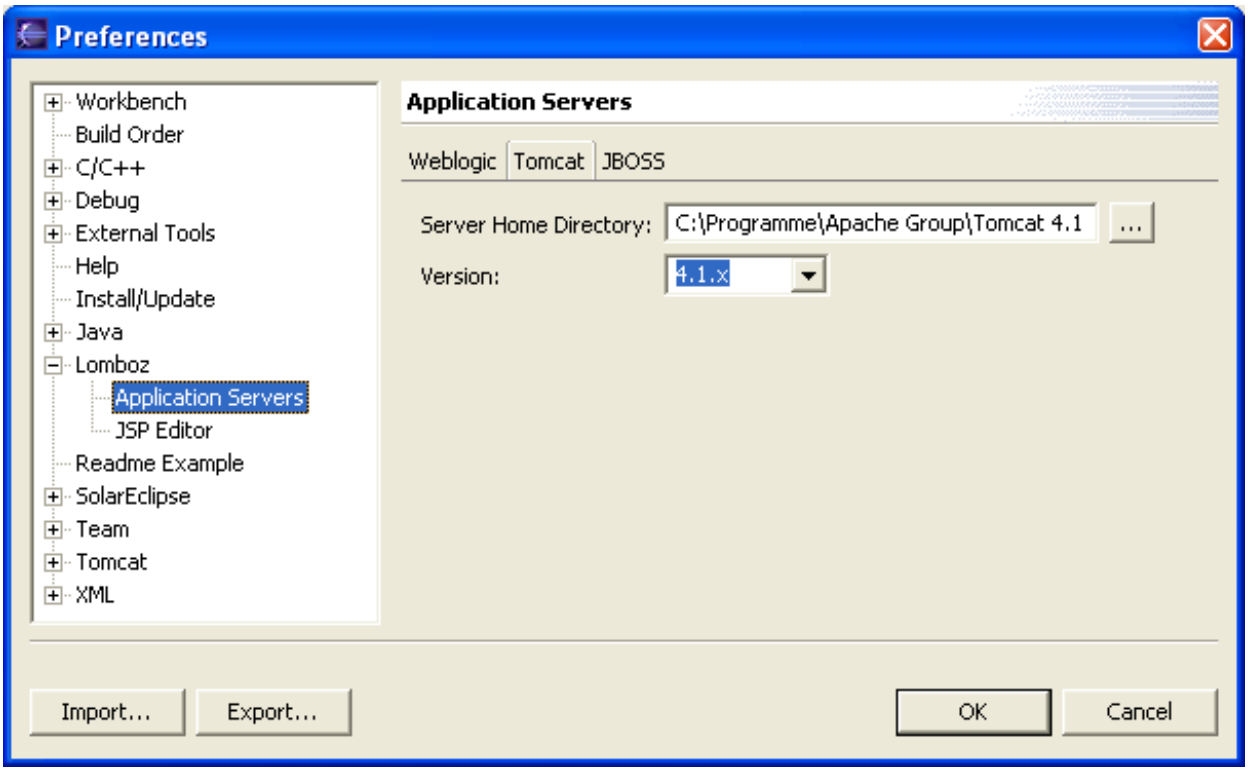
Zunächst wird das ZIP−Archiv in die Verzeichnishierarchie von Eclipse entpackt. Nach dem Neustart von Eclipse läßt sich das Plugin unter Windows / Preferences konfigurieren:

Abbildung 6.5. Konfiguration von Lomboz 1



Zunächst gibt man den Pfad zur `tools.jar` vor. Das ist die Bibliothek aus dem JDK welche den Compiler enthält. Falls man mehrere JDKs installiert hat, sollte man darauf achten, dass der Compiler des JDKs verwendet wird, welches auch für das Projekt eingestellt ist.

Abbildung 6.6. Konfiguration von Lomboz 2

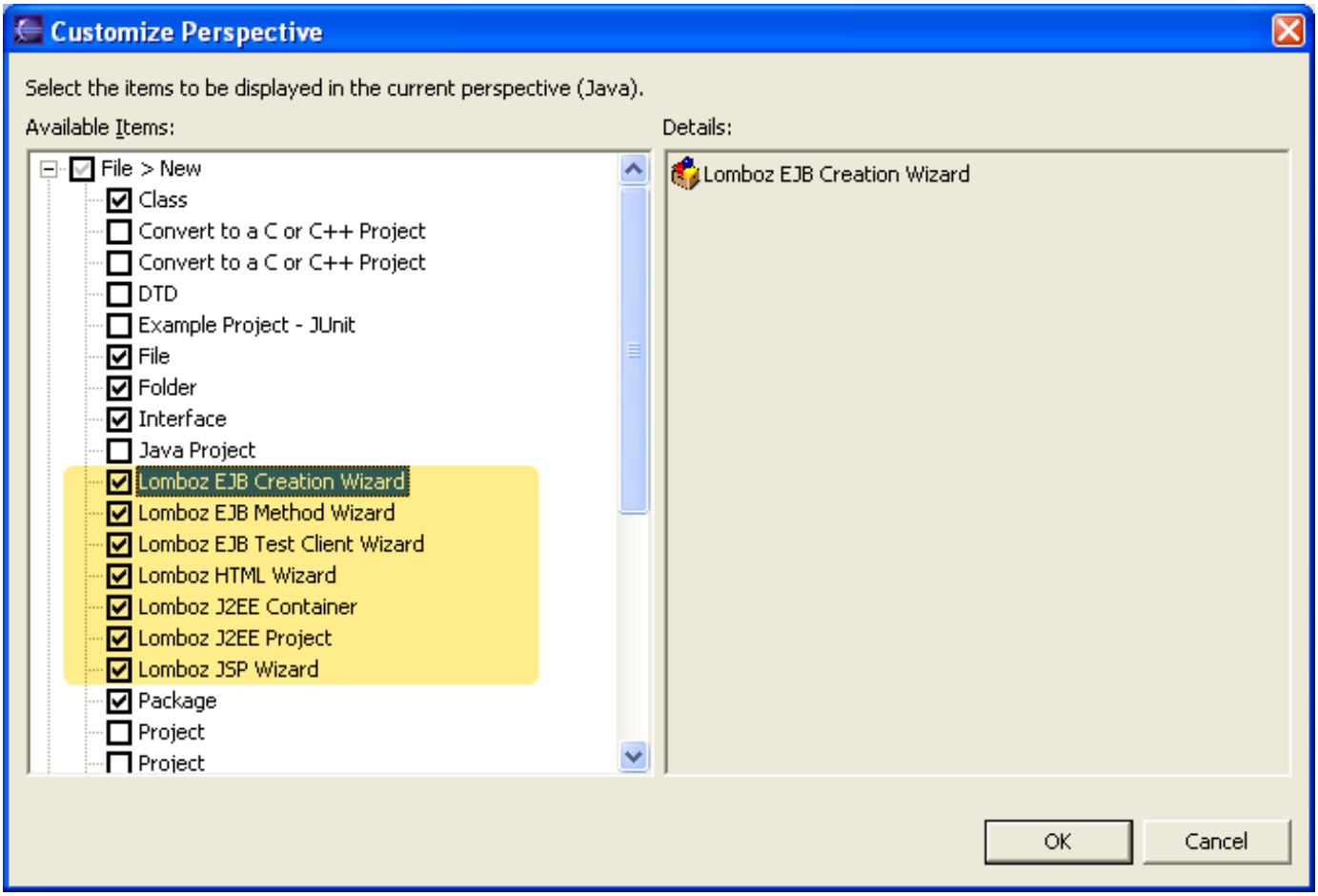


Dann wählt man die passenden Parameter für die Tomcat−Installation.

Die Einstellungen des JSP−Editor sind eher kosmetischer Natur und sollen hier nicht weiter erwähnt werden.

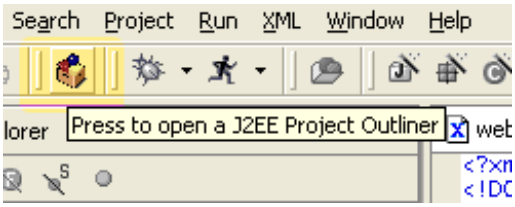
Damit die verschiedenen Views und Actions sichtbar werden, muss man in der Java−Perspective über Customize Perspective (genau wie oben beim Tomcat−Plugin) einige der Lomboz−Optionen aktivieren:

Abbildung 6.7. Konfiguration von Lomboz 3



Entsprechend werden im Zweig Windows / Show View der Lomboz J2EE View aktiviert und im Zweig Other die Lomboz Actions .

Neben den Wizards zur Erstellung neuer Resourcen, kommt in der Taskleiste damit ein neuer Button J2EE Project Builder hinzu.



6.3. Wizards des Lomboz−Plugins

Neben dem Verwalten von J2EE Containern, kann Lomboz noch mehr Unterstützung leisten. Es gibt Wizards zum Erzeugen von diversen Objekten. Ausser den EJB bezogenen sind dies folgende:

- Lomboz J2EE Project: erzeugt die komplette Projektstruktur in einem Rutsch, das Ergebnis sieht dann so ziemlich genauso aus, wie das was im Folgenden vorgestellt wird.
- Lomboz HTML Page: erzeugt ein Gerüst für eine HTML−Seite
- Lomboz J2EE Container: erzeugt einen neuen J2EE (Web) Container (wie oben schon erläutert)
- Lomboz JSP Page: erzeugt eine neue JSP Seite. Neben der Definition der Fehler−Seite, lassen sich hier bereits zu benutzenden Beans, mit Id, Scope usw. Eintragen.
- Lomboz Servlet Wizard: erzeugt den Rumpf einer Servlet−Klasse.
- Lomboz SOAP−Client Wizard: erzeugt eine neue Stub−Klasse mit der man auf einen WebService zugreifen kann.

Ausser dem Lomboz SOAP−Client Wizard werde ich hier nichts benutzen.

---

[3] (W)eb−(A)pplication−A(R)chive

Kapitel 7. Vorbereitungen

Die eigentliche Axis Installation verläuft wie unter http://ws.apache.org/axis/java/install.html beschrieben. Um WebServices jedoch besser testen und debuggen zu können, wähle ich ein etwas anderes Setup.

Das Projekt wird als WebApplication innerhalb von Eclipse angelegt und die Laufzeitumgebung von Axis miteingebunden. So läuft die ganze Anwendung fürs Debuggen in Eclipse und man kann einen Request jederzeit unterbrechen und Schritt für Schritt abarbeiten. Das Output−Verzeichnis wird direkt in den Classpath des Web−Containers (hier verwende ich Tomcat) gelegt, so dass bei Veränderungen kein erneutes Kopieren der Class−Files notwendig ist. Jede Änderung wird von Tomcat erkannt und die betreffende Klasse wird automatisch neu geladen.

Folgende Schritte sind hierzu notwendig:

- J2EE Project anlegen mit Lomboz
- Web−Container erstellen
- Sysdeo−Plugin benutzen, um Applications−Context direkt in Eclipse zu verwenden.
- Axis−Laufzeitumgebung einbinden

7.1. Plugins konfigurieren

Die Plugins von Sysdeo und Lomboz werden wie bei Eclipse Plugins üblich einfach in die Programmhierarchie von Eclipse hinein entpackt. Anschließend sind für beide Plugins ein paar Pfade anzupassen. Da ich Tomcat als Servlet−Container verwende, ist jeweils der Tomcat−Pfad in beiden Plugins einzustellen.

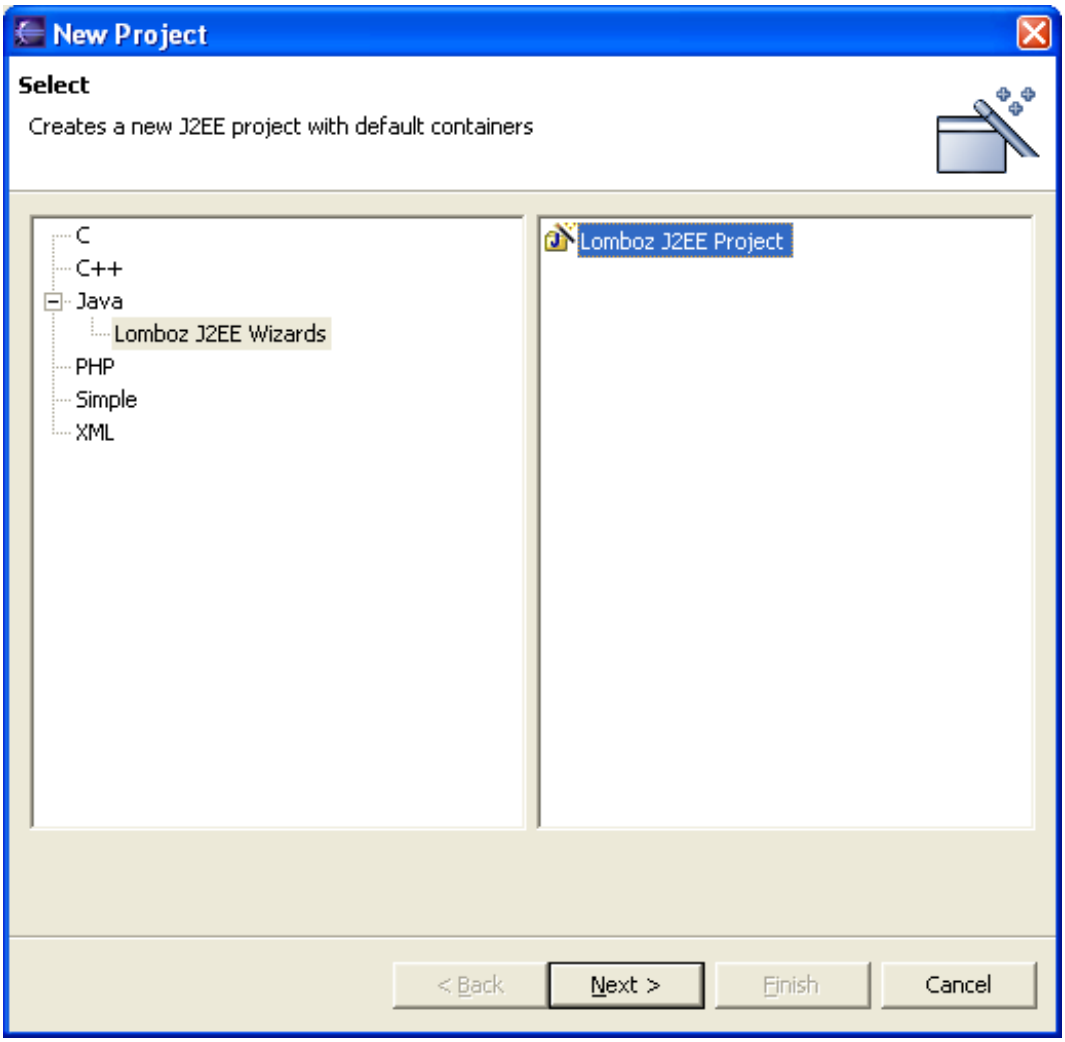
Für das Lomboz−Plugin ist zusätzlich die Java−Perspective anzupassen (siehe Querverweis zu WebApplications).

Zu beachten: Der hier verwendete Tomcat 4.1.29 muss beim Sysdeo−Plugin bereits als Tomcat 5.X eingetragen werden, sonst geht der Start des Servers schief.

7.2. J2EE Projekt anlegen und Web−Container anlegen

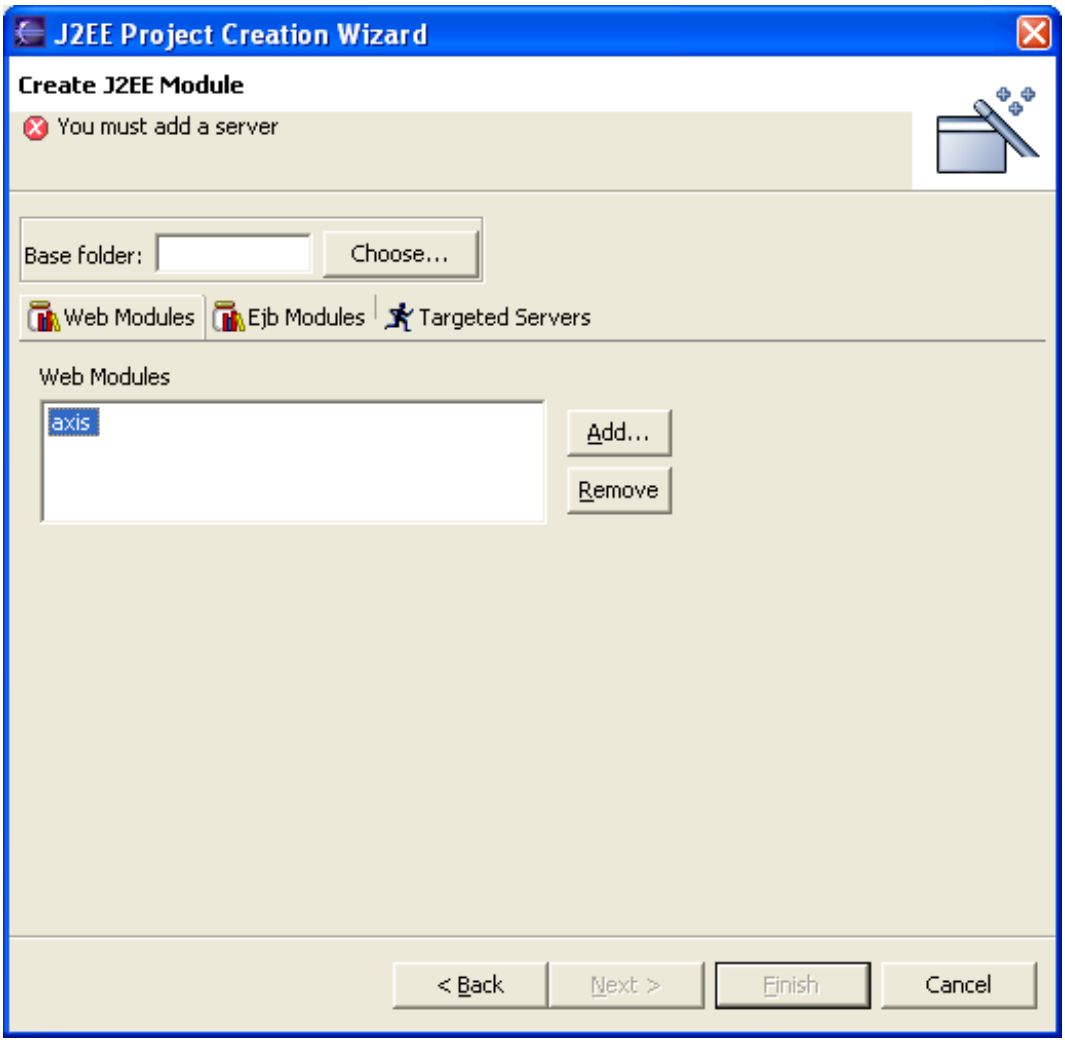
Mit New > Project: Java > J2EE Project wird ein neues Projekt angelegt.

Abbildung 7.1. Neues Projekt mit Lomboz anlegen



Nach Eingabe des Names und der Java−Settings wird in Schritt 4 wird ein neuer Web−Container hinzugefügt und als Target−Server Tomcat 4.1.X gewählt.

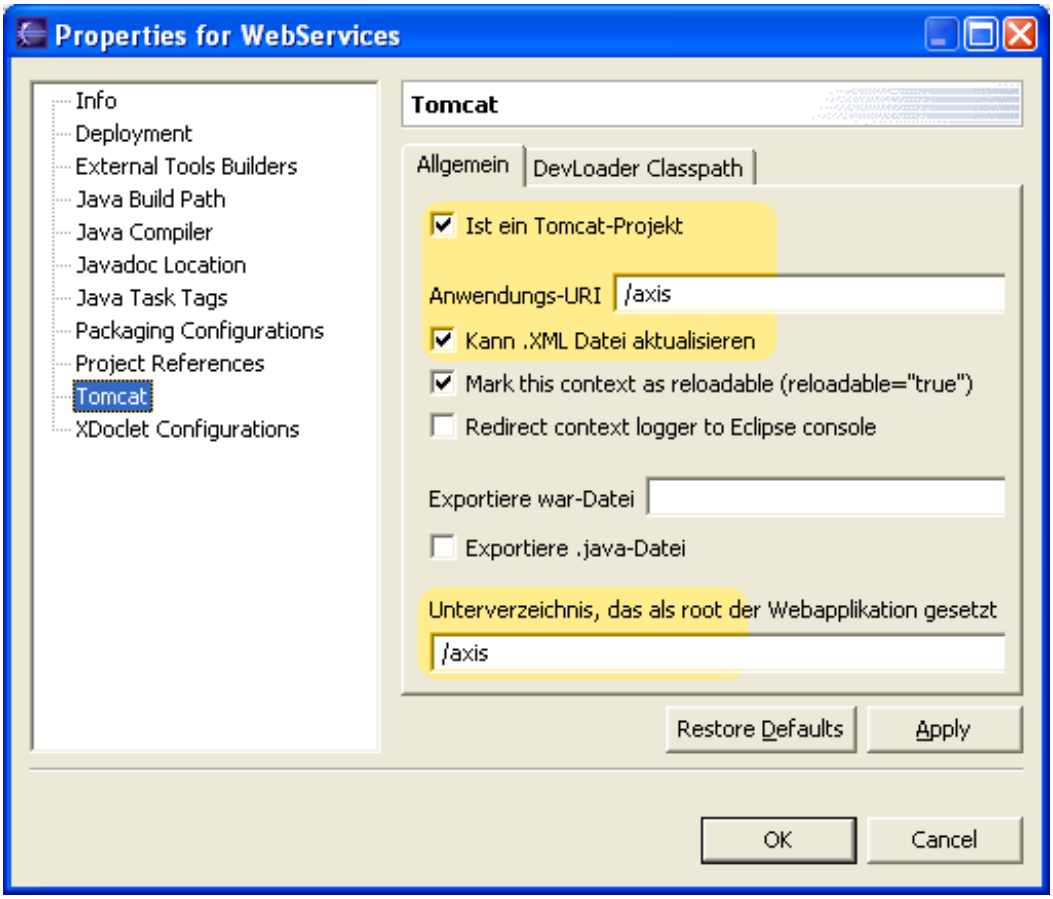
Abbildung 7.2. Web Modul anlegen mit Lomboz



7.3. WebApplication direkt in Eclipse starten

Unter Projekt−Eigenschaften wählt man ist ein Tomcat−Projekt und stellt folgende Pfade ein. So kann später Tomcat direkt gestartet werden und die WebApplication wird direkt im Workspace von Eclipse ausgeführt. Das Kopieren der Class−Files entfällt.

Abbildung 7.3. Projekt−Properties für Sysdeo−Tomcat



Die Anwendungs−Uri bestimmt das Präfix der URI unter der die Application im Browser zu erreichen ist (hier http://localhost:8080/axis/). Das Unterverzeichnis der WebApp−Root (hier ebenfalls axis) gibt an in welchem Projekt−Unterverzeichnis sich der Web−Container befindet.

7.4. Axis integrieren

Die Axis−Installation überschneidet sich etwas mit dem Lomboz−Plugin, das selbst schon eine Axis−Runtime mitbringt. Diese ist allerdings nicht ganz auf dem aktuellen Stand und ausserdem sind nicht alle Bibliotheken dabei. Hinzu kommt noch dass Lomboz die Classpath−Variable AXIS immer auf die mitgebrachten Libraries setzt.

All das umgeht man am einfachsten in dem man die kompletten Axis Bibliotheken aus der Axis−Distribution über die von Lomboz kopiert (copy AXIS_BASE/lib/*.jar ECLIPSE_BASE/plugins/com.objectlearn.jdt.j2ee/axis).

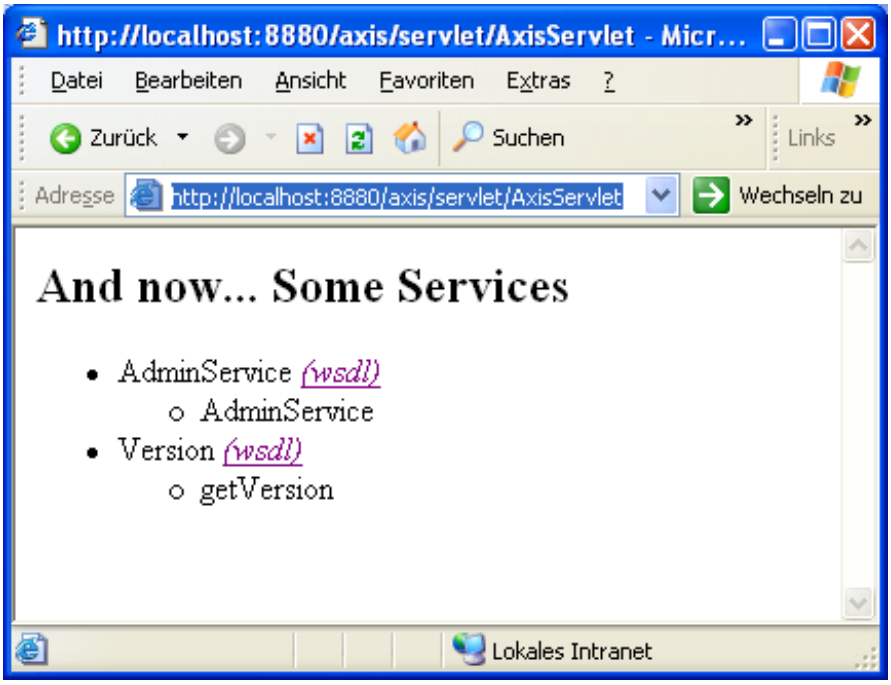
7.5. Axis WebApplication einfügen

Als nächstes kopiert man die Beispiel−WebApplication von Axis in den Workspace (oder importiert die Files mit Eclipse): (copy AXIS_BASE/webapps/* WORKSPACE/<ProjectName>/axis). Die Class−Files, die in der Binary−Distribution von Axis schon kompiliert mitgeliefert werden, kann man wieder löschen (liegen unter WEB−INF/classes).

7.6. Erster Test

Mit den Sysdeo−Plugin kann man nun schon mal Tomcat starten und auf die Axis−WebApplication zugreifen (http://localhost:8080/axis/servlet/AxisServlet). Es sollten zwei WebServices bereits verfügbar sein:

Abbildung 7.4. Liste der deployten WebServices

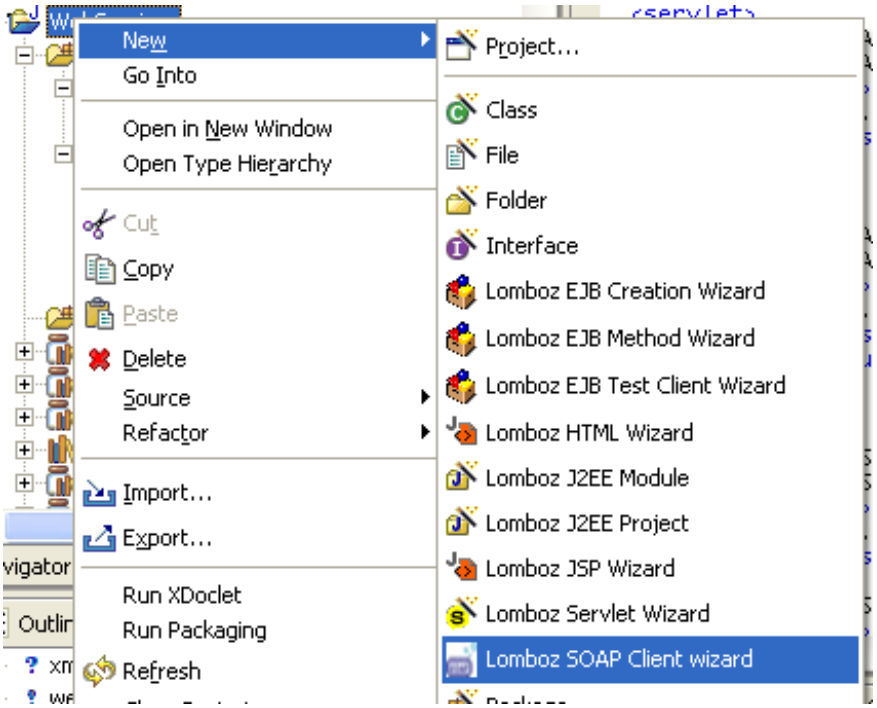


Mit einem Klick auf die wsdl −Links kann man sich auch gleich die zugehörigen WSDL−Beschreibungen anzeigen lassen.

Kapitel 8. WebServices nutzen

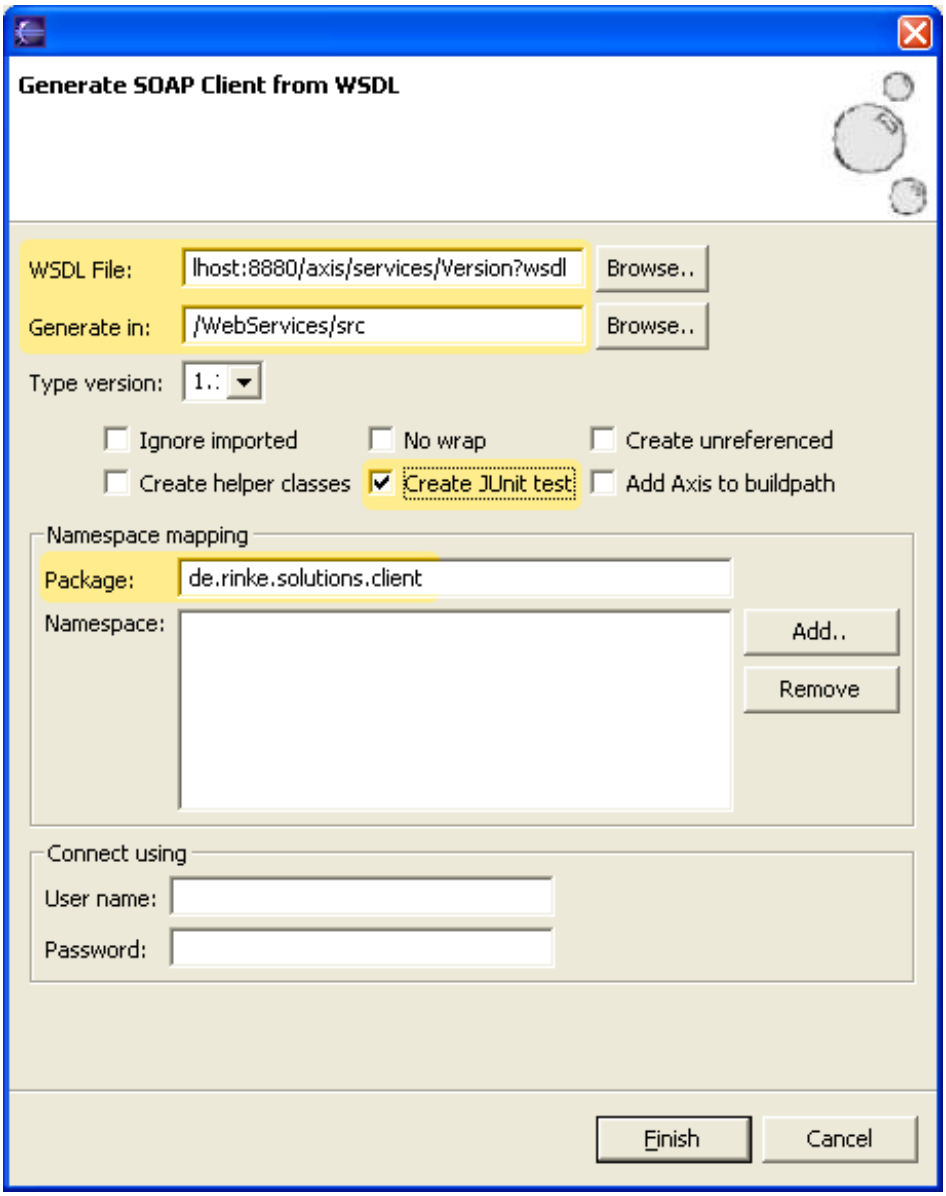
Um von Java aus auf WebServices zugreifen zu können, gibt es mehrere Möglichkeiten. Die einfachste ist ausgehend von einer WSDL−Datei einen Stub erzeugen zu lassen, der den WebService auf der Clientseite repräsentiert. Diesen Stub kann man dann genauso benutzen wie jede andere Java−Klasse. Dass dahinter ein WebService die Arbeit verrichtet merkt man gar nicht[4]. Das Axis−Framework liefert zu diesem Zweck ein Tool mit Namen WSDL2Java was solche Stubs generieren kann. Da das direkte Aufrufen doch eher umständlich ist, lassen wir das das Lomboz−Plugin erledigen:

Abbildung 8.1. Aufruf des WebService Wizards in Lomboz



Man kann entweder die WSDL−Datei vom Filesystem lesen, dann muss man sie zuvor mit dem Browser mit Speichern unter ... gesichert haben. Alternativ kann man auch direkt eine URL eingeben: http://localhost:8080/axis/services/Version?wsdl

Abbildung 8.2. WebService Wizards in Lomboz



Wenn man zusätzlich create Unittest ankreuzt, wird auch gleich ein Unittest erzeugt, der zeigt wie man die generierten Klassen einsetzt. Der TestCase verlangt allerdings junit.jar auf dem Classpath, so dass man die Projekteigenschaften entsprechend anpassen muss.

Anschließend finden sich im angegebenen Package vier neue Quellcode−Files, die zusammen den Client für den Zugriff auf den Version −WebService ermöglichen:

- Version.java ist das Interface das den WebService beschreibt.
- VersionService.java ist das Interface welches einen Stub liefert (die Factory).
- VersionSoapBindingStub.java ist der Stub, der Version implementiert.
- VersionServiceLocator.java ist die Implementierung der Factory

Als weiteres fünftes File wird noch ein TestCase generiert, der den WebService aufruft. Um den TestCase auszuführen markiert man einfach die Datei VersionServiceTestCase.java und wählt Run / Run as ... / JUnit TestCase.

---

[4] Stimmt nicht ganz, da durch den RemoteCall natürlich bestimmte Exceptions potentiell nicht zu vermeiden sind.

Kapitel 9. WebServices erstellen

9.1. WebServices mit JWS−Dateien

Die einfachste Möglichkeit einen WebService mit Axis zu erstellen geht so: Man nimmt eine einfache Java−Klasse die eine oder mehrere öffentliche Methoden zur Verfügung stellt und benennt sie von *.java nach *.jws um.

Anschließend compiliert Axis diese Java−Klasse beim ersten Zugriff und deployed sie als WebService. Ein explizites Deployment ist gar nicht notwendig.

Die EchoHeaders.jws die bei Axis mit ausgeliefert wird, ist ein Beispiel dafür. Die kompilierte Klasse landet in WEB−INF/jwsClasses. Der Mechanismus ist dem von JSP−Seiten sehr ähnlich. Will man solche WebServices mit Eclipse debuggen, geht das allerdings nicht ganz ohne Probleme. Ausserdem ist die Steuerung der Inbetriebnahme über einen Deployment−Descriptor nicht vorgesehen.

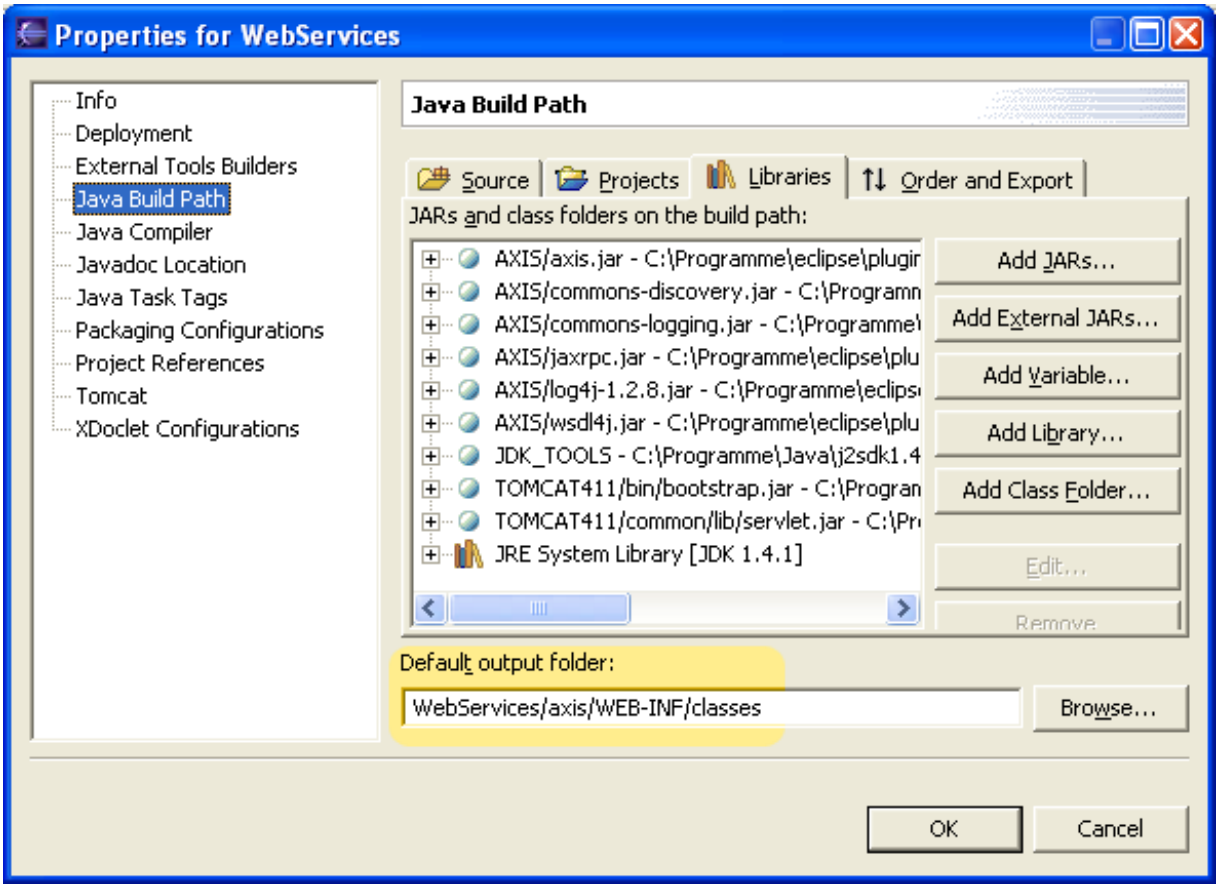
9.2. WebServices als normale Java−Klassen

Man kann einen WebService mit Axis als ganz einfache Java−Klasse (plain old java object oder POJO) realisieren. Alles was zusätzlich gebraucht wird, ist ein Deployment−Descriptor. Damit man diesen aber nicht immer getrennt von den Java−Quellcodes pflegen muss, werde ich ihn hier mit XDoclet−Kommentaren erzeugen.

Neben der eigentlichen Java−Klasse braucht man hierzu noch zwei Dinge: ein Ant−Script und ein XDoclet−Template.

Damit Eclipse die kompilierten Class−Files immer gleich an die richtige Stelle schreibt, muss in den Projekteigenschaften das Ausgabeverzeichnis auf den Web−Container geändert werden (siehe Beispiel−Projekt):

Abbildung 9.1. Ändern des Ausgabeverzeichnises



9.2.1. Ein WebService

Im beigefügten Beispiel−Projekt ist ein primitiver WebService realisiert:

```
/**
 * @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) Mit diesen XDoclet−Tags wird die Klasse als WebService deklariert

(2) @axis.method markiert die Methode als exportierte WebService−Methode. Diese Methode ist damit von aussen aufrufbar.

9.2.2. Das XDoclet−Template

Im Beispiel−Projekt ist ein spezielles Template für XDoclet enthalten, welches einen Deployment−Descriptor für Axis erzeugt. Es wertet die im Kapitel 10 beschriebenen Tags aus und erzeugt so ein 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" paran
     </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. Das Build-Script

Das Ant-Script steuert die Erzeugung des Deployment-Descriptor mit XDoclet und kann anschließend gleich das Deployment in das laufende Axis Anwendung hinein vornehmen. Hierzu werden aus der Axis-Distrubution die Ant-Tasks in axis-ant.jar eingebunden und mit der Admin-Task das Deployment vorgenommen. Das Script wird noch parametriert über build.properties, welche alle lokalen Einstellungen enthält.

```
<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) Diese Pfad-Definition legt den Classpath für XDoclet fest. Um mit Ant XDoclet-Tasks zu benutzen müssen die XDoclet-Bibliotheken verfügbar sein. Die build.property xdocletlib.dir legt somit das Verzeichnis fest, in dem die Xdoclet-Bibliotheken zu finden sind.

(2) Diese Pfad-Definition legt den Classpath für Axis fest. Um mit Ant Axis-Tasks nutzen zu können müssen die Axis-Bibliotheken verfügbar sein. Hier verwende ich einfach einen Verweis auf das WEB-INF/lib-Verzeichnis, weil dort in diesem Projekt sowieso alle Axis-Bibliotheken installiert sein müssen.

(3) Dieses Taskdef-Tag definiert die Template-Task für XDoclet.

(4) Das templatedoclet -Tag führt das XDoclet-Template aus, um den Deployment-Descriptor zu erzeugen.

(5) Das Fileset legt fest, dass alle Java-Dateien im src-Verzeichnis betrachtet werden.

(6) Das axis-deploy Target führt das Deployment aus. Es hängt von axisdoclet ab, d.h. ggf. wird der Deployment-Descriptor vorher neu erzeugt.

(7) Dieses Taskdef-Tag definiert alle Ant-Tasks die von Axis unterstützt werden (siehe Axis-Dokumentation).

(8) Durch die axis-admin Task wird das Deployment durchgeführt. Hierzu müssen in den build.properties alle Parameter wie target.port, target.server usw. Richtig eingestellt sein. Ausserdem muss der Server natürlich laufen und das AdminServlet muss aktiviert sein (siehe web.xml im Beispiel-Projekt).

Welche WebServices gerade aktiviert sind kann jederzeit mit einem erneuten Aufruf der Axis-Seite /axis/servlet/AxisServlet abgefragt werden. Das Servlet verwendet übrigens ein XML-File server-config.wsdd in dem alle deployten Services aufgeführt sind. Wir ein neuer Service in Betrieb genommen, dann passt Axis diese Datei entsprechend an.

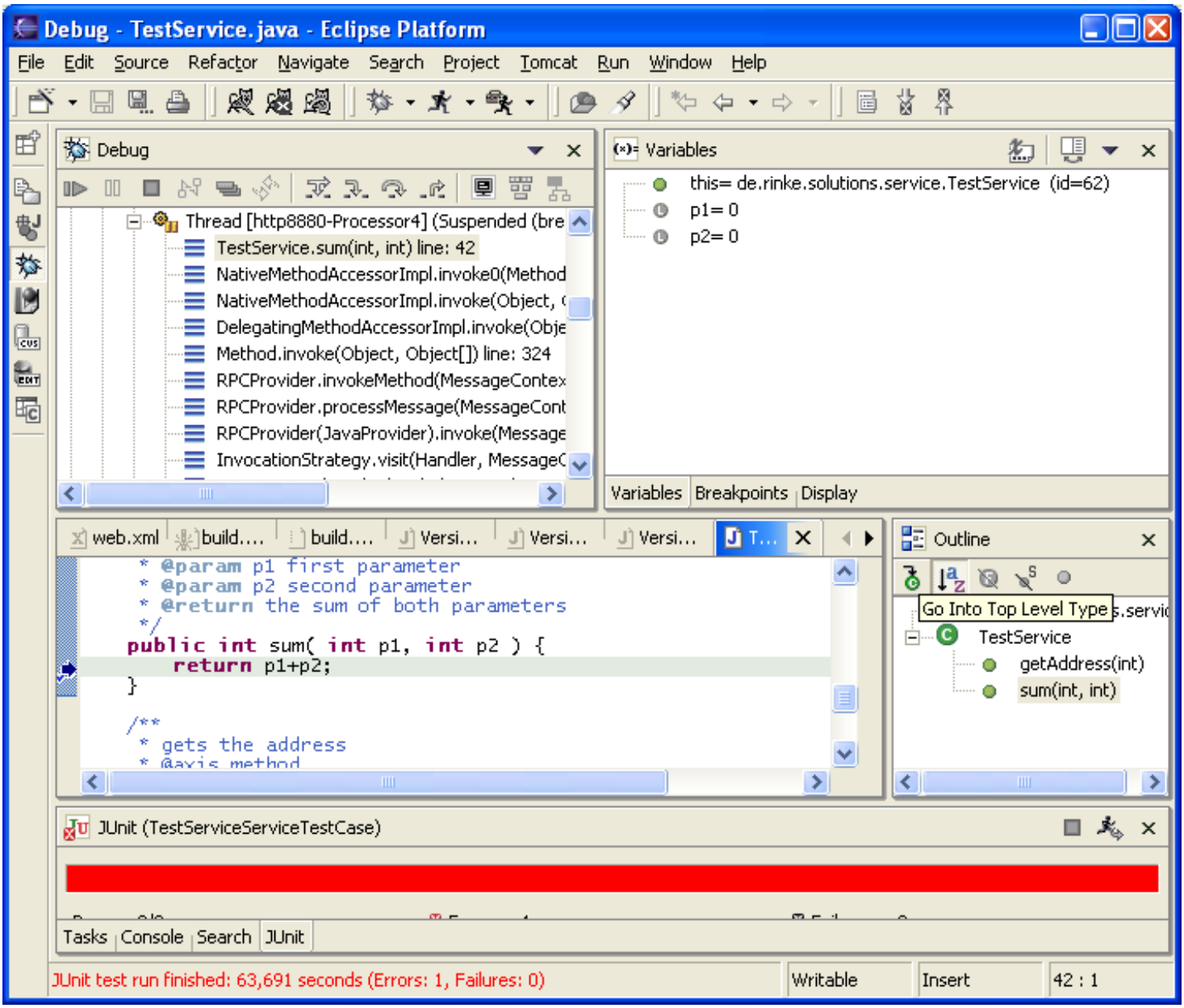
9.3. WebServices als Enterprise Java Beans

Wird in einer späteren Version nachgereicht.

9.4. WebServices debuggen

Da der WebService mitsamt Tomcat komplett in Eclipse läuft kann man einfach in der implementierenden Klasse einen Breakpoint setzen.

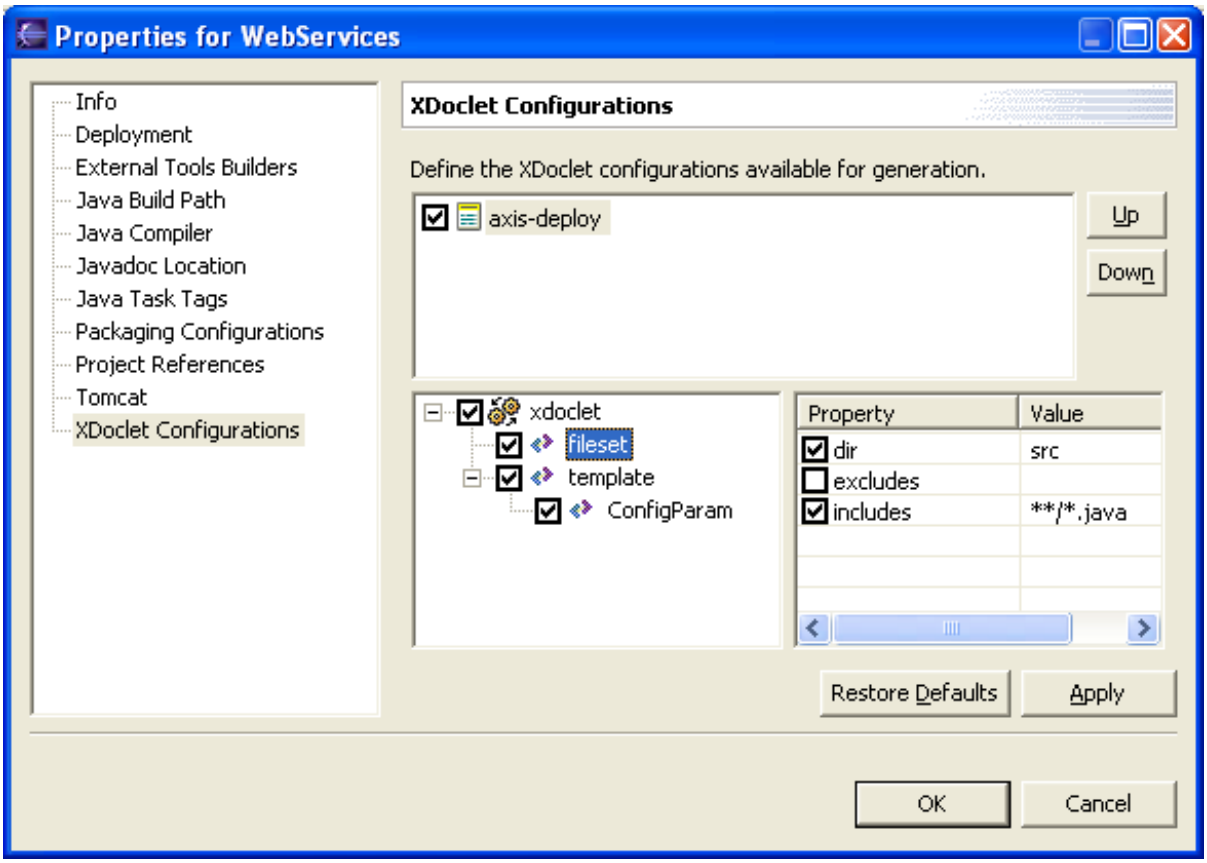
Abbildung 9.2. WebService im Debugger



9.5. XDoclet mit JBOSS−IDE

Das Plugin JBOSS−IDE erlaubt eine alternative Steuerung von XDoclet. Statt Doclets in einem Ant−Script zu definieren und aufzurufen, kann eine grafische Oberfläche benutzt werden.

Abbildung 9.3. Grafische XDoclet−Konfiguration



Über den Abschnitt XDoclet−Configurations in den Projekteigenschaften kann man sich XDoclet−Aufgaben zurechtklicken . Man definiert letztlich genau dasselbe was auch in dem schon vorgestellten Ant−Script steht, nur eben mit Hilfe dieses Dialogfensters. Für Einsteiger ist diese Methode evtl. besser geeignet, weil man sich um bestimmte Details, wie den Xdoclet−Path (wo die XDoclet−JARs liegen) keine Gedanken machen muss. Ausserdem werden alle möglichen Varianten direkt angezeigt, so dass man nicht überlegen muss, wie dieser oder jener Parameter genau heißt.

Die erzeugte Konfiguration kann man sich im Beispiel−Projekt ansehen, wenn man in Eclipse das Dialogfenster erneut aufruft. Alternativ wirft man einen Blick in `xdoclet-build.xml` (im Wurzelverzeichnis des Projekts), das ist die Datei in der das JBOSS−IDE−Plugin diese Einstellungen speichert.

Um XDoclet mit dieser Konfiguration zu starten, gibt es nun sogar einen Menüeintrag im Kontextmenü des Projekts.

9.6. Neue XDoclet−Version

Während des Schreibens dieses Artikels erschien eine neue XDoclet Version 1.2 (kein Beta2). Diese enthält auch eine neue Subtask axisdeploy , die unterhalb des EJB−Doclets aufgerufen werden kann. Leider war die Dokumentation noch nicht auf dem aktuellen Stand, so dass diese zunächst nur in dem Manning Titel XDoclet in Action

nachzulesen war. Mit dieser Subtask wird ebenfalls ein Deployment−Descriptor erzeugt, so dass sich diese Funktionalität mit der hier vorgestellten überschneidet.

Kapitel 10. Neue XDoclet Tags

Der Deployment−Descriptor von Axis beschreibt neben den verantwortlichen Klassennamen, eine Methodenliste, einen Request−Flow usw. (siehe Axis−Dokumentation). Um all diese Attribute zu unterstützen sind einige neue Tags erforderlich, die alle Axis−Attribute beschreiben. Hinzu kommt ein Template für XDoclet, welches die Tags verarbeitet.

10.1. Class−Level−Tags

10.1.1. axis.service

Deklariert die Klasse als WebService.

Attribute	Beschreibung	Default / Optional
name	Der Name des WebServices	− / verbindlich
scope	Der Gültigkeitsbereich des WebServices. Mögliche Werte: request, session, application. Request−Scope erzeugt für jeden Request eine neue Instanz der Java−Klasse, Application verwendet ein Singleton, das alle Requests bedient und Session erzeugt eine neue Instanz für jeden neuen Client, der Sessions unterstützt.	request / optional
urn	Der Name des WebServices unter dem er von aussen aufgerufen werden kann. Wenn nichts angegeben wird, dann wird der Name der Klasse verwendet	ClassName / optional
enable−remote−admin	Wenn true gesetzt, kann der Service remote administriert werden. Voraussetzung ist allerdings die globale Aktivierung des RemoteAdmin−Features in Axis.	false / optional
include−all	Wenn true gesetzt werden alle öffentlichen Methoden der Klasse als WebService exportiert.	false / optional

10.1.2. axis.handler

Deklariert die Klasse als Request−Handler. Sie muss von `org.apache.axis.handlers.BasicHandler` abgeleitet sein.

Attribute	Beschreibung	Default / Optional
name	Der Name des Handlers	− / verbindlich

10.1.3. axis.useHandler

Deklariert einen Handler, den die WebService−Klasse benutzten soll. XDoclet fügt damit im Deployment−Descriptor ein Request−Flow Element ein, welches den/die Handler einbindet.

Attribute	Beschreibung	Default / Optional
name	Der Namen des zu benutzenden Handlers	− / verbindlich

10.1.4. axis.parameter

Deklariert einen Parameter, den die Handlerklasse erfragen kann.

Attribute	Beschreibung	Default / Optional

name	Der Name des Parameters	− / verbindlich
value	Der Wert des Parameters	− / verbindlich

10.2. Method−Level−Tags

10.2.1. axis.method

Deklariert eine Methode als WebService−Methode. Für den Fall, dass auf Klassenebene nicht include−all=true als Attribut angegeben wurde, kann so jede Methode einzeln als WebService−Methode deklariert werden.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.

Kapitel 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.

Kapitel 12. Download WebServices mit Java, Axis, XDoclet und Eclipse

- Beispiel−Projekt
- PDF
- HTML
- Microsoft Word
- Windows Help
- Plaintext
- HTML (eine Datei)
- XML (DocBook)
- OpenOffice
- Eclipse−Plugin
- Build Framework (um selbst mit DocBook solche Artikel zu schreiben)

Kapitel 13. Anhang

13.1. Online−Referenzen

13.1.1. Eclipse Plugin Verzeichnisse

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

13.1.2. Andere Online−Artikel zu Eclipse

- 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. Andere Online−Artikel zu Axis

- Die SOAP Engine Apache AXIS
- Apache Axis: Architektur und Erweiterbarkeit

13.1.4. Allgemeine Online Resourcen zu Eclipse

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

13.1.5. Weitere Plugins für WebServices mit Eclipse

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