Mirzaonline©

Working with JBoss and Eclipse

Trademark of JBoss Inc (www.jboss.org)

Trademark of Eclipse open source community (www.eclipse.org)

Preface

When during the summer 2005 I decided to learn more about JBoss(as one of the leading application servers in the J2EE market), I realized that documentation available on the official JBoss website was quite poor by its structure. The basic problem with JBoss documentation is that it doesn't show step by step tutorial for deployment of J2EE applications. What it provides is a bunch of scripts and files of the Duke's Bank application, that after executing automatically create needed tables, files, bytecode and deploy all modules at once. It never goes into detail how to deploy each particular module separately. Most of other websites don't go further from the "Hello World" examples. Probably the most unusual part of the JBoss tutorial is that it doesn't show how to integrate JBoss with Eclipse by using JBoss-IDE. I find it very important because it makes development and deployment easier. A lot of other people on the Internet complained about the same problem. One of the explanations could be that JBoss makes money on training but not on software license, since JBoss server is "Open Source" and free for download. I am not professional and my tutorial aims to introduce people into JBoss from slightly different angle than standard JBoss-tutorial, giving some basic idea how to deploy the most fundamental components of J2EE by using Eclipse and JBoss together. Of course, it is not meant to be complete coverage of neither JBoss nor Eclipse. Eclipse is explained only in the context of JBoss and J2EE. There is much more Eclipse than that! Since English is not my native language, some errors are inevitable.

The summer of 2005
Mirza Jahic

Index

PART 1

Application Server –JBoss

BEA WebLogic, JBoss, Web Sphere, Sun Server are probably the most widely used Java application servers in the world. Web Logic is the most popular commercial server and JBoss is the most popular open source server. This document is about JBoss and basic work with this server.

Introduction to JBoss architecture

The most important part of every Java Server is a container. It is not very difficult to imagine what container could be. It is simply a set of classes (just like anything else in Java) that manage particular object(s). That object could be any J2EE component. It could be EJB, it could be a servlet , and it could be JSP, Java Bean, JMS or any other component. Each component runs in its own container that manages that component. This is what I've found on the Internet about J2EE containers:

container *A Java run-time environment for enterprise beans. A container, which runs on an Enterprise JavaBeans server, manages the life cycles of enterprise bean objects, coordinates distributed transactions, and implements object security.*

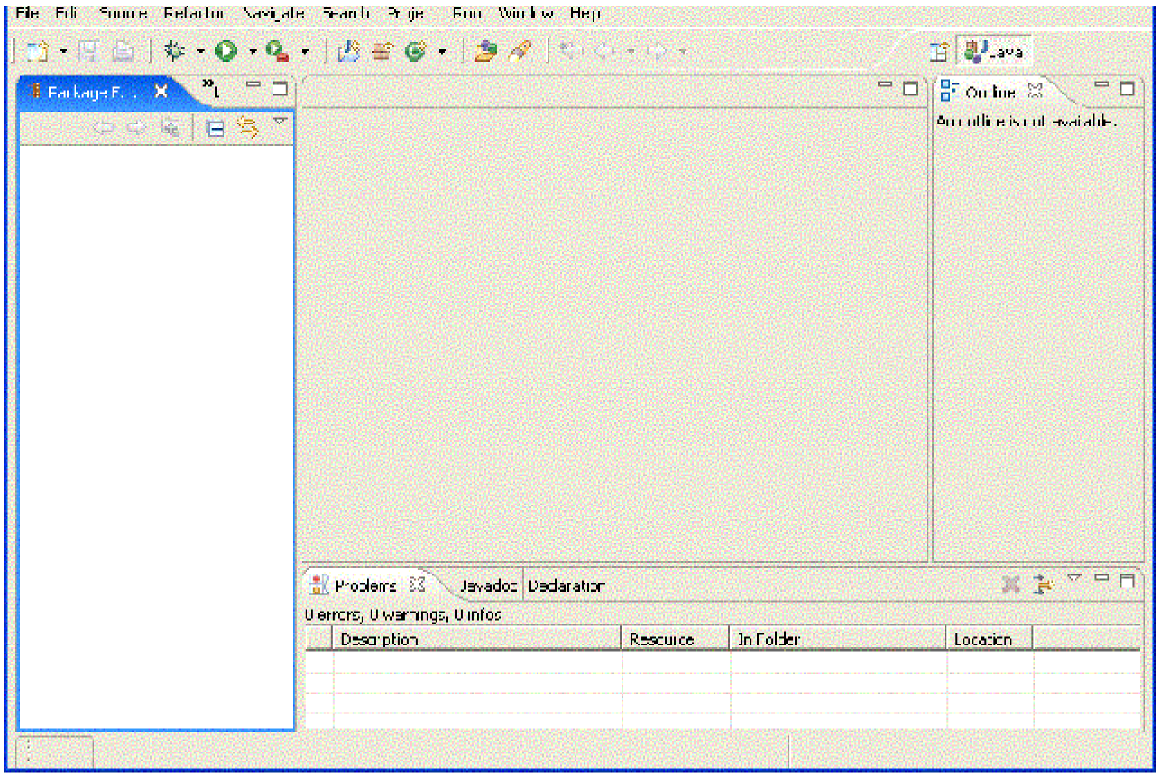
> *In J2EE, an entity that provides life-cycle management, security, deployment, and run-time services to components. (Sun) Each type of container (EJB, Web, JSP, servlet, applet, and application client) also provides component-specific services.*

Tomcat 5.5 is the most popular container and it is embedded into many application servers, among them into JBoss. It is important to say that JBoss is a wrapper around that container and provides interaction with the container through its beans, called "MBeans". MBean is probably the most important part of that wrapper together with container (microkernel). The most important folders for an average user are **bin** (where you can start and stop server), **config** (where you can change configuration of the server by using different XML-files), **deploy** (where you can deploy your application). The last 2 are parts of the bigger folder called **server**. It provides all needed files and folders for 3 different configurations of JBoss (**all** (maximal), **minimal** and **default (**standard**)**). Default is the most widely used configuration. It is used for deployment of the most important J2EE standard components. More information about JBoss architecture at: **www.jboss.org** or **www.jboss.com**.

Eclipse

The story about JBoss wouldn't be complete without saying few words about Eclipse. Eclipse is an Integrated Development Environment (IDE). It is one of the most popular, pure Java IDEs that provides a lot of plugins. These plugins are modules that make Eclipse more powerful. These plugins are Java plugin, C++ plugin (called CDT), COBOL plugin (used to be popular), UML plugin, Ant, Junit , XDoclet and JBOSS-IDE plugin. There are many more but the last one is the most important because it provides integration with JBoss server. Lomboz is also kind of JBOSS-Eclipse plugin but it will not be a subject in this document. Eclipse can be downloaded for free at : www.eclipse.org.

JBoss-IDE provides ability to start and stop server, deploy and undeploy application from the JBoss, generate descriptor files and also view logs. Probably there are even more features that didn't come to my mind or I just simply don't know about.


File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package E...
Outline
Problems
Javadoc
Declaration

An outline is not available.

0 errors, 0 warnings, 0 infos

	Description	Resource	In Folder	Location

Figure 1: Eclipse in Action (The first run)

After downloading Eclipse from the web, the good thing to do is simply to create a batch file that will start Eclipse on click (Eclipse comes zipped without Windows-installer). So instead of every time typing something like this: **eclipse.exe -showlocation -vm C:\jdk1.5_04\bin\javaw.exe -vmargs -Xmx256M ,** you can simply save this line into a notepad and than call it ***start .bat***. By the way this line shows the path for my virtual machine (**C:\jdk1.5_04\bin\javaw.exe)** so that Eclipse can run it and also defines amount of RAM needed for Eclipse (**Xmx256M).**

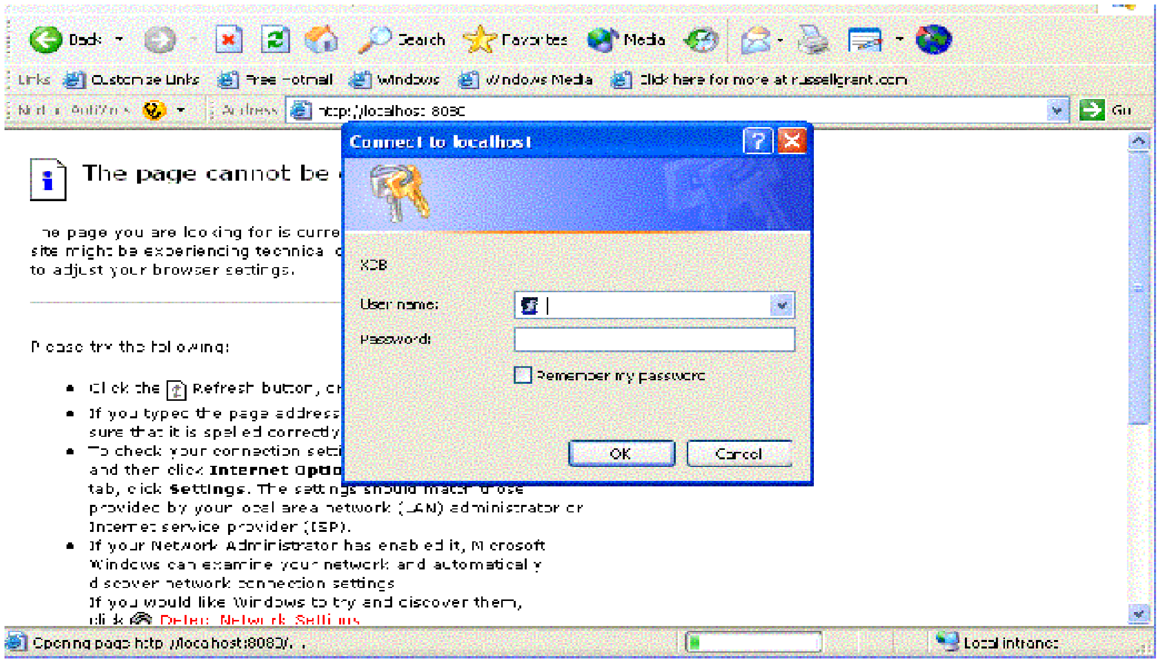
Without going into details about Eclipse and what it can do, leave it for now and let's go back to JBoss server.

Basic configuration and usage of JBoss

Before integrating JBoss with Eclipse it is useful to see what can we do with JBoss alone.
Some the most basic commands of JBoss that can be found in the **bin** directory are:

-run .bat	**-Run the server**
-shutdown.bat	**-Shutdown the server**
-twiddle.bat	**-Connects to the remote server at port 1099**

It is very simple to use these commands by just typing in the command line something like let say **run** and hitting the ENTER is enough to start the server. But before doing that it is important to configure server to listens at free port. JBoss by default listens at the port 8080 and sometimes that port can be taken by some other server/listener. The easiest way to check it is simply to try to access console (JMX-console or Web-console) of the JBoss through the browser. Something like: **localhost: 8080.** Let see my case!



The page cannot be

The page you are looking for is curre
site might be experiencing technica
to adjust your browser settings.

Please try the following:

- Click the Refresh button, or
- If you typed the page address
  sure that it is spelled correctly
- To check your connection setti
  and then click **Internet Optio**
  tab, click **Settings**. The settings should match those
  provided by your local area network (LAN) administrator or
  Internet service provider (ISP).
- If your Network Administrator has enabled it, Microsoft
  Windows can examine your network and automatically
  discover network connection settings.
  If you would like Windows to try and discover them,
  click Detect Network Settings

Connect to localhost

XDB

User name:

Password:

Remember my password

OK Cancel

Opening page http://localhost:8080/...

Local intranet

Figure 2: Checking the free ports of the free world at localhost: 8080

In my case port 8080 is taken by another listener, namely ORACLE9i that I installed some time ago. It uses listener called XDBA. This is a kind of server that enables to ORACLE databases to be accessed over the Internet. Well in this case I don't have any choice but to find some other port for my JBoss. I guess that port 8081 would be OK.

Changing the port in JBoss

Port configuration must be done in 2 different files. The first file is called **server.xml** (**default\deploy\jbossweb-tomcat55.sar\server.xml**) and here portion of the code that contains port number looks like this:

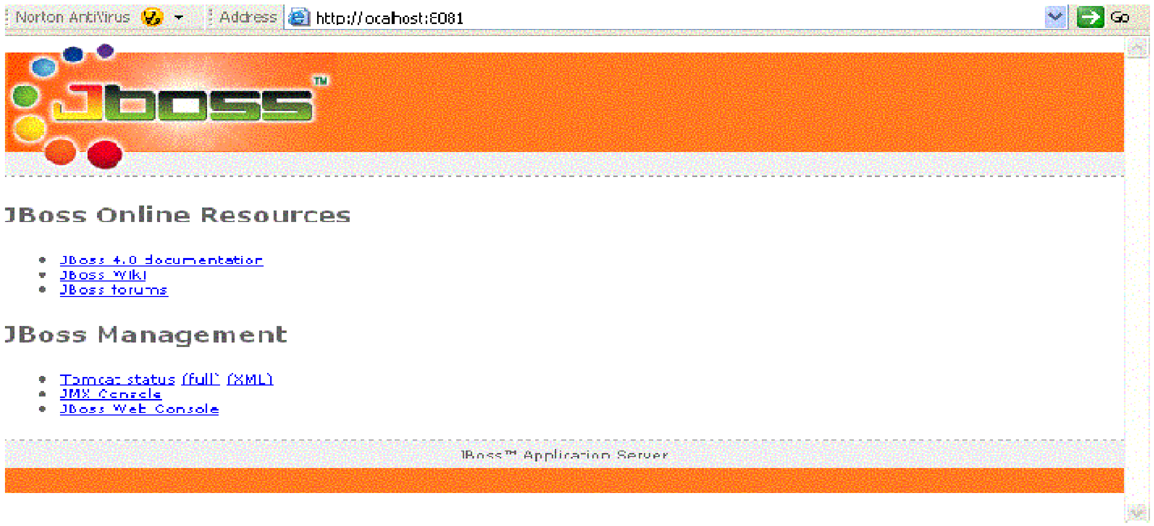
```
<Connector port="8080" address="${jboss.bind.address}" maxThreads="250" strategy="ms"
maxHttpHeaderSize="8192" emptySessionPath="true" enableLookups="false"
redirectPort="8443" acceptCount="100" connectionTimeout="20000"
disableUploadTimeout="true" />
```

Port must be changed to 8081 in my case. The other file that contains port number is **called jboss-service.xml** (**default\deploy\http-invoker.sar\META-INF\jboss-service.xml**) .

```
<attribute name="InvokerName">jboss:service=Naming</attribute>
<attribute name="InvokerURLPrefix">http://</attribute>
<attribute name="InvokerURLSuffix">:8080/invoker/readonly/JMXInvokerServlet</attribute>
```

<attribute name="UseHostName">true</attribute>

Here also port number must be changed to 8081. There was also chance to "kill" the ORACLE services by going into Control Panel/Services but this would stop use of ORACLE database. After this change and after running **run.bat** JMX-console is here!



JBoss Online Resources

- JBoss 4.0 documentation
- JBoss Wiki
- JBoss forums

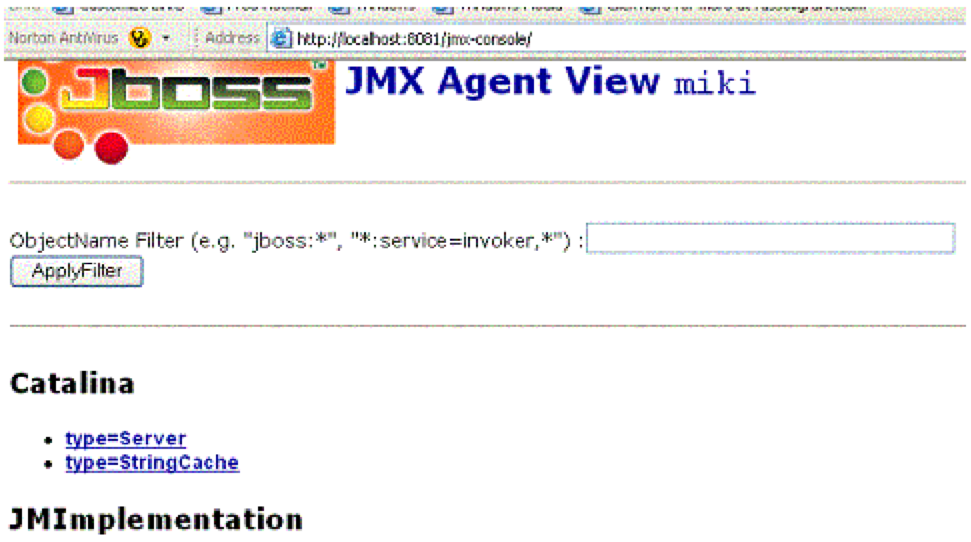
JBoss Management

- Tomcat status (full) (XML)
- JMX Console
- JBoss Web Console

JBoss™ Application Server

Figure 3: JBoss console is up and running!

Now when the port is changed to 8081 and JBoss is accessible through the browser it is important to say that JBoss provides 2 consoles. The one is called JMX-console and it is more widely used (I guess since it is simpler). JMX-console is simpler to use, it contains only a servlet as an interface (Figure 4) while Web-console is a mixture of an applet and servlet (Figure 5).



JMX Agent View miki

ObjectName Filter (e.g. "jboss:*", "*:service=invoker,*") :

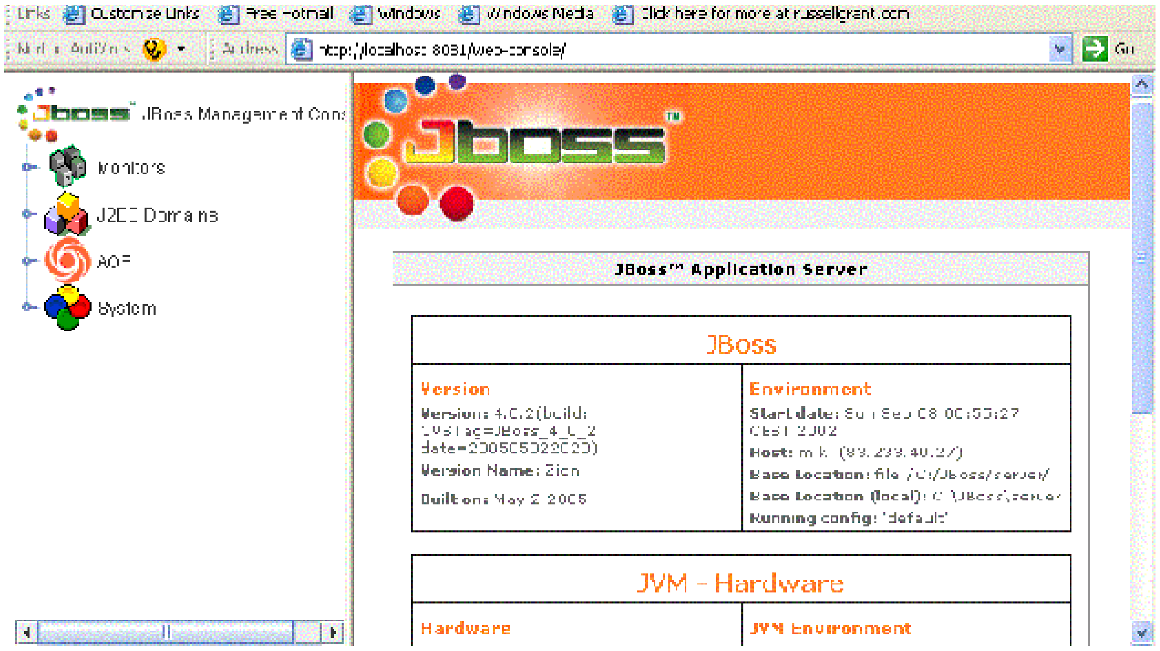
ApplyFilter

Catalina

- type=Server
- type=StringCache

JMImplementation

Figure 4: JMX-console



JBoss™ Application Server

JBoss

Version
Version: 4.0.2(build: CVSTag=JBoss_4_0_2 date=200505022023)
Version Name: Zion
Built on: May 2 2005

Environment
Start date: Sun Sep 08 00:55:27 CEST 2002
Host: m k (83.233.40.27)
Base Location: file:/C:/Jboss/server/
Base Location (local): C:\JBoss\server
Running config: 'default'

JVM - Hardware

Hardware

JVM Environment

Figure 5: Web-console

All features can be controlled through any of these consoles (like starting up database Hypersonic). One more important thing to say is that access and security to these 2 consoles can be controlled, so that password and username must be provided (if needed).

Security issues with JMX-console and Web-console

Accessing the application server over the Internet is easy. Anyone who is connected to the Internet can access your JBoss server and make changes. This is very dangerous scenario and that's why JBoss provides security features for the JMX-console and the Web-console. Authentication is done with a password and username.

In order to setup this security feature for JMX-console and Web-console, three XML files must be changed (commented out). The first one is called "**jboss-web.xml**" and you can find it in "**default\deploy\jmx-console.war\WEB-INF**" (**Or/and deploy\management\console-mgr.sar\web-console.war\WEB-INF\jboss-web.xml for** web-console). This file looks like this:

```
<jboss-web>
<!---
 Uncomment the security-domain to enable security. You will
      need to edit the htmladaptor login configuration to setup the
      login modules used to authentication users.
      <security-domain>java:/jaas/jmx-console</security-domain>
```

```
-->
</jboss-web>
```

This file implements Java Authentication and Authorization Service (JAAS). JAAS is a standard J2EE feature implemented by JBoss and by all other enterprise servers. The second file is called "**login-config.xml**" and it can be found at "**server\default\conf**". This file is used for authentication and authorization of all J2EE components, those that belong to JBoss and those that belong to other deployed applications. By default this portion of code is always uncommented for JMX-console and Web-console. This is how that portion of code looks like (for jmx-console):

```
<application-policy name="jmx-console">
<authentication>
<login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule" flag="required">
<module-option name="usersProperties">props/jmx-console-users.properties</module-option>
<module-option name="rolesProperties">props/jmx-console-roles.properties</module-option>
</login-module>
</authentication>
</application-policy>
```

Similar code is used for the web-console, only name is different. This portion of code tells us that authentication is done by using "**UsersRolesLoginModule"** and 2 files, the first one is called "**users.properties**" and it keeps information about users (username=password). The other file keeps information about roles of the users. Admin role is the default one. By default information in the "**users.properties**" contains "admin=admin" and "**roles.properties**" keeps information about roles of the users, by default "admin=JBossAdmin" (JBoss administrator called admin). New users can be simply added into these files, in my case I will add "mirza=123" and "mirza=JBossAdmin". This has to be done in files for JMX and Web consoles. JMX files can be found at " "**server\default\conf\props**" and Web-console can be found at "**default\deploy\management\console-mgr.sar\web-console.war\WEB-INF\classes**".

At the end, the third file must also be modified. That file is called "**web.xml**" and in both cases, it can be found together with "**jboss-web.xml"** in the same folder**.**

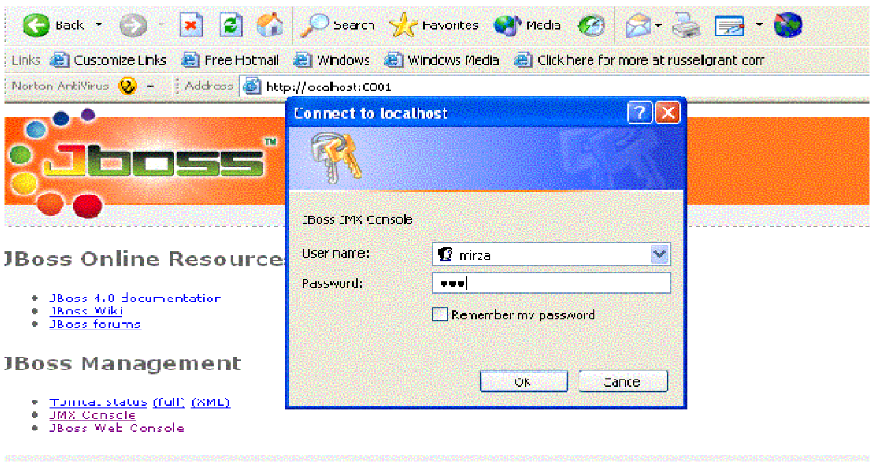
The security portion of code that must be commented out is this:

```
<security-constraint>
  <web-resource-collection>
  <web-resource-name>HtmlAdaptor</web-resource-name>
  <description>An example security config that only allows users with the
  role JBossAdmin to access the HTML JMX console web application
  </description>
  <url-pattern>/*</url-pattern>
  <http-method>GET</http-method>
  <http-method>POST</http-method>
```

```
</web-resource-collection>
<auth-constraint>
<role-name>JBossAdmin</role-name>
</auth-constraint>
</security-constraint>
```

One file for Web-console and the one for JMX-console must be commented out.

After making these changes and perhaps adding new users (in my case) it will be needed



JBoss JMX Console

User name: mirza

Password: •••

Remember my password

Figure 6: Access restricted to JMX and Web consoles!

to restart JBoss, in order changes to take a place. New configuration should ask for the password and username, after clicking jmx or web consoles. Access is restricted.

Integration with Eclipse

JBoss-IDE is the most popular plugin for Eclipse. It is used to enable control of JBoss server through the Eclipse. Installation of this plugin goes in several steps and it is quite easy.

1. **Download plugin from the sourceforge.net or go to Help-> Software Updates->Find and Install.**

I decided to download the latest plugin for Eclipse 3.1, which is called JBoss-IDE1.52M and it comes also with support for EJB 3.0. The zipped plugin contains 2 folders, the one is called "plugins" and the other one is called "features". These jar files must be copied together to the rest of Eclipse files.

2. **The second step (after copying two folders) is to start Eclipse and do following configurations.**

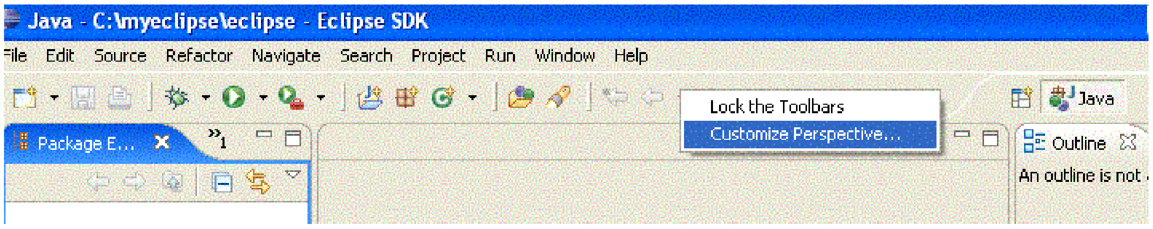
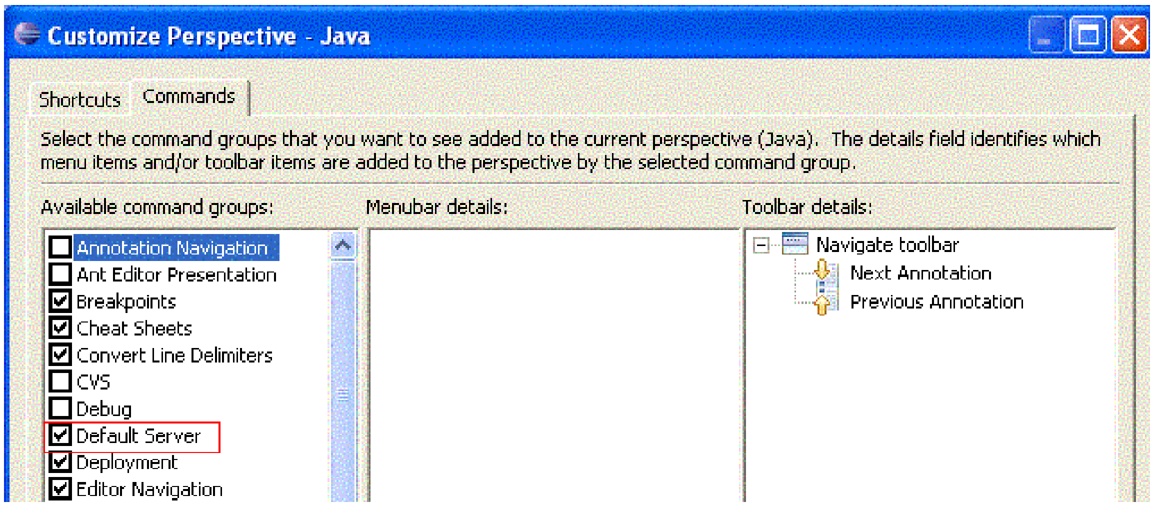

Java - C:\myeclipse\eclipse - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Lock the Toolbars
Customize Perspective...
Java
Package E...
Outline
An outline is not

Figure 7: Click with the right button of the mouse and choose "Customize Perspective"

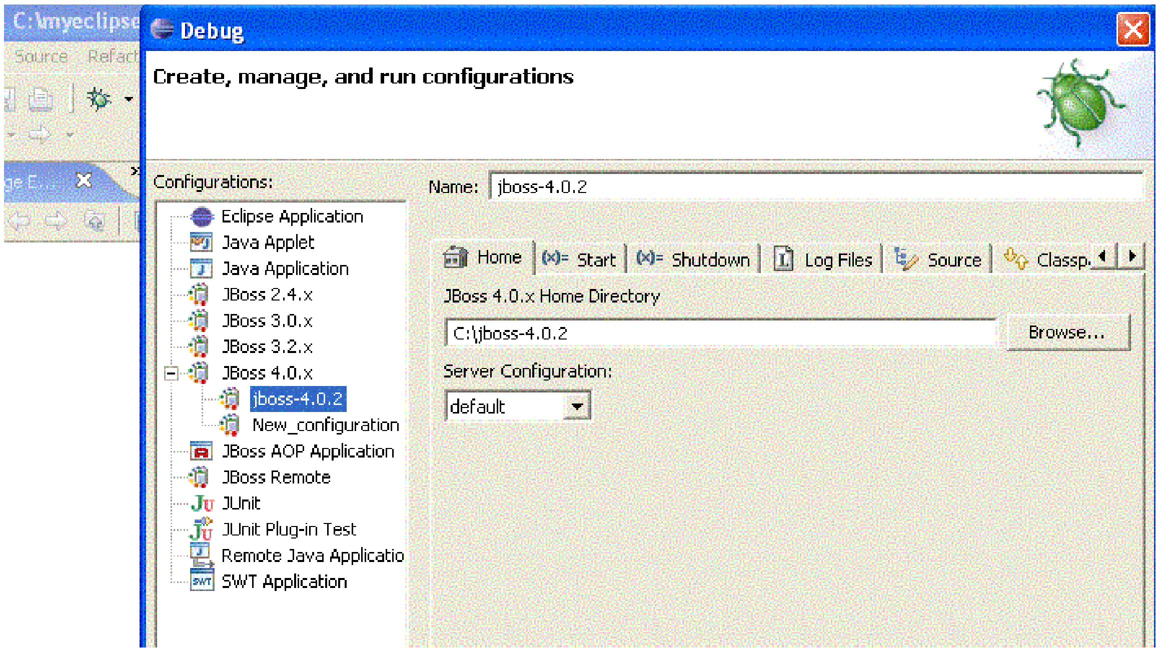


Select the command groups that you want to see added to the current perspective (Java). The details field identifies which menu items and/or toolbar items are added to the perspective by the selected command group.

- [ ] Annotation Navigation
- [ ] Ant Editor Presentation
- [x] Breakpoints
- [x] Cheat Sheets
- [x] Convert Line Delimiters
- [ ] CVS
- [ ] Debug
- [x] Default Server
- [x] Deployment
- [x] Editor Navigation

Figure 8: Click the tab "Commands" and select Default Server

3. **Choose Run from the menu and then choose submenu "Debug"**



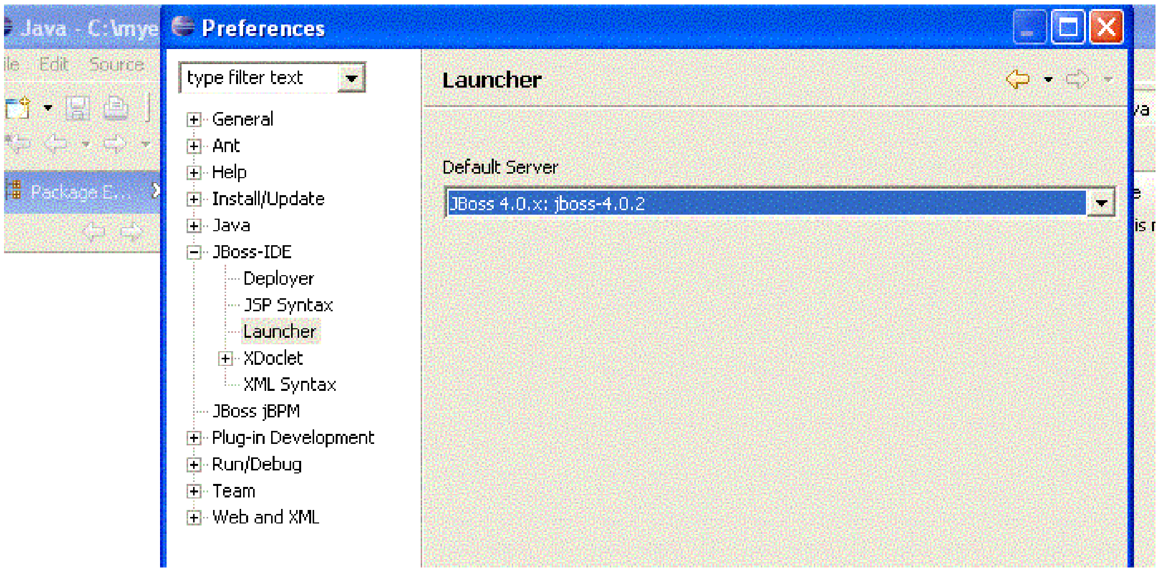
Name: jboss-4.0.2

JBoss 4.0.x Home Directory: C:\jboss-4.0.2

Server Configuration: default

Figure 9: Insert name of the JBoss server in Name and Browse to the location of JBoss

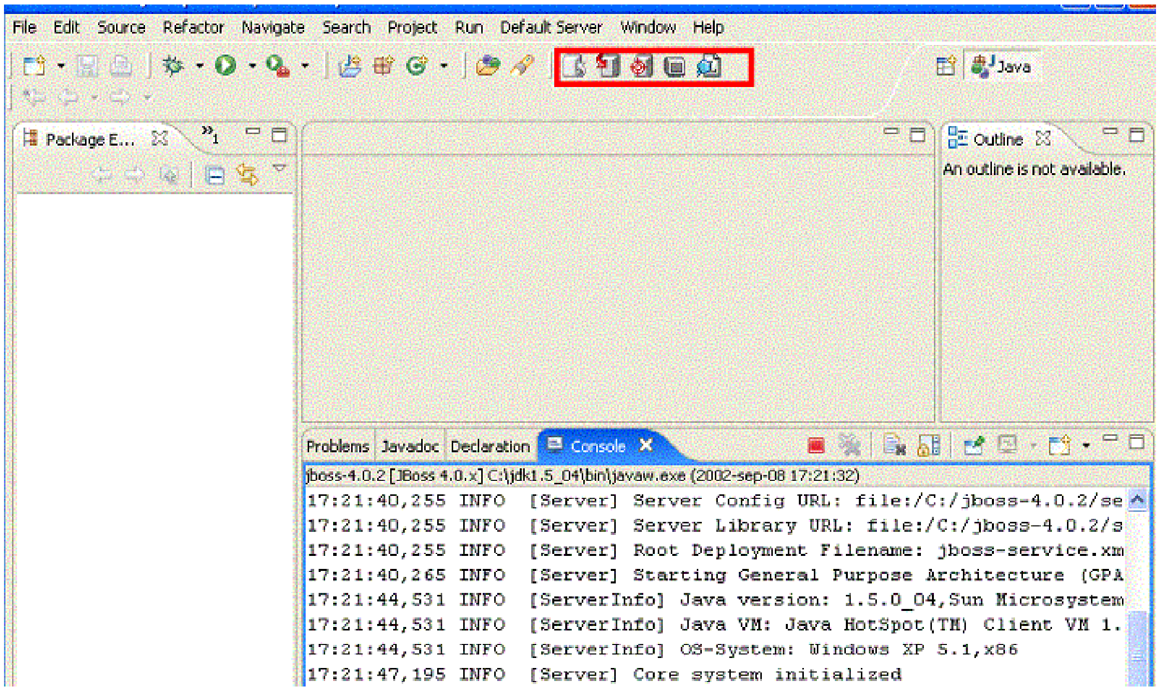
4. **Go to Windows-> Preferences select Launcher and then JBoss version from the drop-down box.**



Default Server: JBoss 4.0.x: jboss-4.0.2

Figure 10: Select the Default Server from the drop-down box

5. **The plugin is now ready to use and by clicking the Start in the menu JBoss will start from Eclipse!**


File Edit Source Refactor Navigate Search Project Run Default Server Window Help
Java
Package E...
Outline
An outline is not available.
Problems Javadoc Declaration Console

```
jboss-4.0.2 [JBoss 4.0.x] C:\jdk1.5_04\bin\javaw.exe (2002-sep-08 17:21:32)
17:21:40,255 INFO  [Server] Server Config URL: file:/C:/jboss-4.0.2/se
17:21:40,255 INFO  [Server] Server Library URL: file:/C:/jboss-4.0.2/s
17:21:40,255 INFO  [Server] Root Deployment Filename: jboss-service.xm
17:21:40,265 INFO  [Server] Starting General Purpose Architecture (GPA
17:21:44,531 INFO  [ServerInfo] Java version: 1.5.0_04,Sun Microsystem
17:21:44,531 INFO  [ServerInfo] Java VM: Java HotSpot(TM) Client VM 1.
17:21:44,531 INFO  [ServerInfo] OS-System: Windows XP 5.1,x86
17:21:47,195 INFO  [Server] Core system initialized
```

Figure 11: JBoss has been launched from the Eclipse!

That was all when it comes to installing the JBoss-IDE and integrating Eclipse and JBoss. Now we can work with JBoss through the Eclipse, including deployment, compilation Xdoclets etc. The next and the last step in our configuration is to configure database. Database must be changed from Hypersonic to ORACLE9I but that is in the next chapter.

Configuration of Database

JBoss comes together with the database called "Hypersonic". It is simple database and it is suitable only for testing and development but far from the real world needs. The basic problem with this database inability to listen and inability to be accessed from the Internet in the same way the ORACLE or MySQL can be.

Changing the database is not very difficult but it is not trivial either. The basic problem is that JMS and EJB services are dependent on Hypersonic database (by default). That means that we have to update several files in order to remove Hypersonic and replace it with ORACLE9i.

For EJB-service datasource must be changed from DefaultDS to **XAOracleDS**. This should be done under the **name** attribute. The portion of code looks like this:

```
 <!-- A persistence policy that persistes timers to a database -->
  <mbean code="org.jboss.ejb.txtimer.DatabasePersistencePolicy"
name="jboss.ejb:service=EJBTimerService,persistencePolicy=database">
    <!-- DataSource JNDI name -->
    <depends optional-attribute-
name="DataSource">jboss.jca:service=DataSourceBinding,name=XAOracleDS</d
epends>
    <!-- The plugin that handles database persistence -->
    <attribute
name="DatabasePersistencePlugin">org.jboss.ejb.txtimer.GeneralPurposeDatabas
ePersistencePlugin</attribute>
  </mbean>
```

If we want to use JMS we have to make some changes here too. Even if I'm not going to run any JMS examples I will make these changes. In the directory "**jms**" at "**\server\default\deploy\jms**" there are two Hypersonic specific files. The first one is "**hsqldb-jdbc-state-service.xml**" and the second one is "**hsqldb-jdbc2-service.xml**". The second one should be replaced by **oracle-jdbc2-service.xml**. The first one should be updated. DataSource should be changed to ORACLE specific, name = **XAOracleDS.**

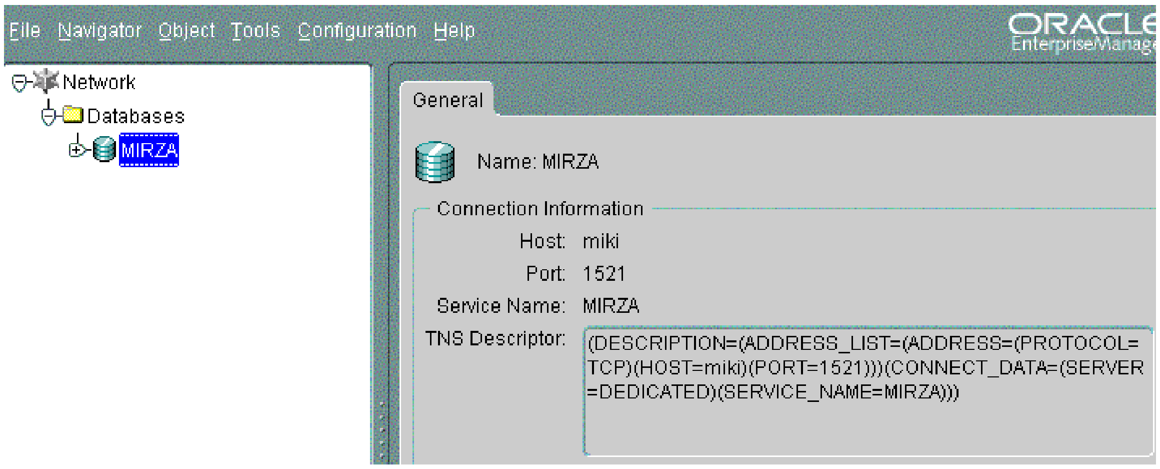
```
<depends optional-attribute-name="ConnectionManager">jboss.jca:service=DataSourceBinding,name=
XAOracleDS </depends>
```

These files can be found in “**JBoss\docs\examples\jms**”.

The next file that has to be updated is “Security Domain for JBossMQ”. This domain can be found in **login-config.xm** file under the “**JBoss\server\default\conf**” directory. The default DataSource should use ORACLE name =” **XAOracleDS”.**

```
<!-- Security domain for JBossMQ -->
   <application-policy name = "jbossmq">
     <authentication>
       <login-module code =
"org.jboss.security.auth.spi.DatabaseServerLoginModule"
          flag = "required">
          <module-option name = "unauthenticatedIdentity">guest</module-option>
          <module-option name = "dsJndiName">java:/ XAOracleDS </module-
option>
          <module-option name = "principalsQuery">SELECT PASSWD FROM
JMS_USERS WHERE USERID=?</module-option>
          <module-option name = "rolesQuery">SELECT ROLEID, 'Roles' FROM
JMS_ROLES WHERE USERID=?</module-option>
       </login-module>
     </authentication>
   </application-policy>
```

The final dependency is to change the file in the “deploy” directory. The Hypersonic specific file must be removed (**hsqldb-ds.xml**) and new file called **oracle-xa-ds.xml** must be placed in the “deploy” folder. But before doing that, information in the oracle-xa-ds.xml must be changed. It is database-specific and can be found in the ORACLE database. In my example information looks like this:



Name: MIRZA

Connection Information

Host: miki

Port: 1521

Service Name: MIRZA

TNS Descriptor: (DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=miki)(PORT=1521)))(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=MIRZA)))

Figure 12: ORACLE information about host, port and database.

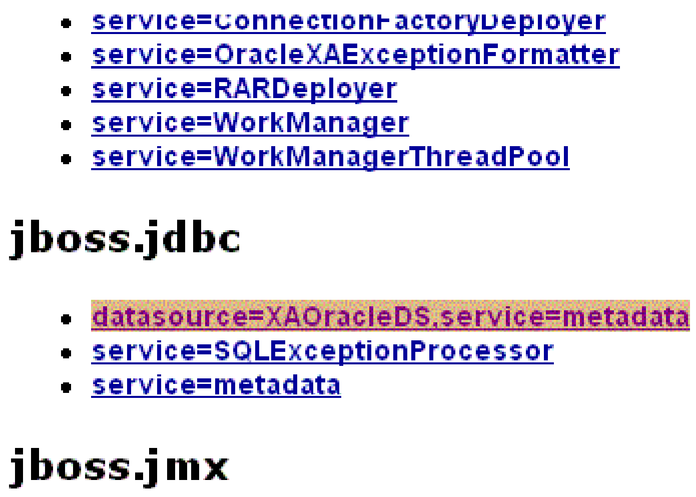
This information should be mapped to **oracle-xa-ds.xml** file and looks like this:

```
<xa-datasource>
    <jndi-name>XAOracleDS</jndi-name>
    <track-connection-by-tx/>
    <isSameRM-override-value>false</isSameRM-override-value>
    <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-
datasource-class>
    <xa-datasource-property
name="URL">jdbc:oracle:thin:@miki:1521:MIRZA</xa-datasource-property>
    <xa-datasource-property name="User">MIRZAJAHIC</xa-datasource-
property>
    <xa-datasource-property name="Password">123456789</xa-datasource-
property>
    <!-- Uses the pingDatabase method to check a connection is still
valid before handing it out from the pool -->
    <!--valid-connection-checker-class-
name>org.jboss.resource.adapter.jdbc.vendor.OracleValidConnectionChecke
r</valid-connection-checker-class-name-->
    <!-- Checks the Oracle error codes and messages for fatal errors --
>
    <exception-sorter-class-
name>org.jboss.resource.adapter.jdbc.vendor.OracleExceptionSorter</exce
ption-sorter-class-name>
    <!-- Oracles XA datasource cannot reuse a connection outside a
transaction once enlisted in a global transaction and vice-versa -->
    <no-tx-separate-pools/>

      <!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml
(optional) -->
      <metadata>
         <type-mapping>Oracle9i</type-mapping>
      </metadata>
  </xa-datasource>
```

The last thing you have to do is to change the DataSource in **uuid.key-generator.sar** file that can be found in “deploy” directory. Since I’m not going to use it (for key-generating) I will simply delete it.

This file (oracle-xa-ds-xml) can be found at "**docs\examples\jca**". After these changes have been made the server should start and the file should be placed into "deploy" directory. After deploying this file the server should bind the datasource and message on the console should be displayed "`19:22:25,683 INFO [WrapperDataSourceService] Bound connection factory for resource adapter for ConnectionManager 'jboss.jca: service=DataSourceBinding, name=XAOracleDS to JNDI name 'java:XAOracleDS'`".

The best way to control if everything went OK is to log in the JMX-console and then to see if the new DataSource has been registered.



- service=ConnectionFactoryDeployer
- service=OracleXAExceptionFormatter
- service=RARDeployer
- service=WorkManager
- service=WorkManagerThreadPool

jboss.jdbc

- datasource=XAOracleDS,service=metadata
- service=SQLExceptionProcessor
- service=metadata

jboss.jmx

Figure 13: Checking the datasource in the JMX-console

If everything went fine we can now delete Hypersonic database from the JBoss server. Hypersonic database can be found in "**server/default/data**" directory.

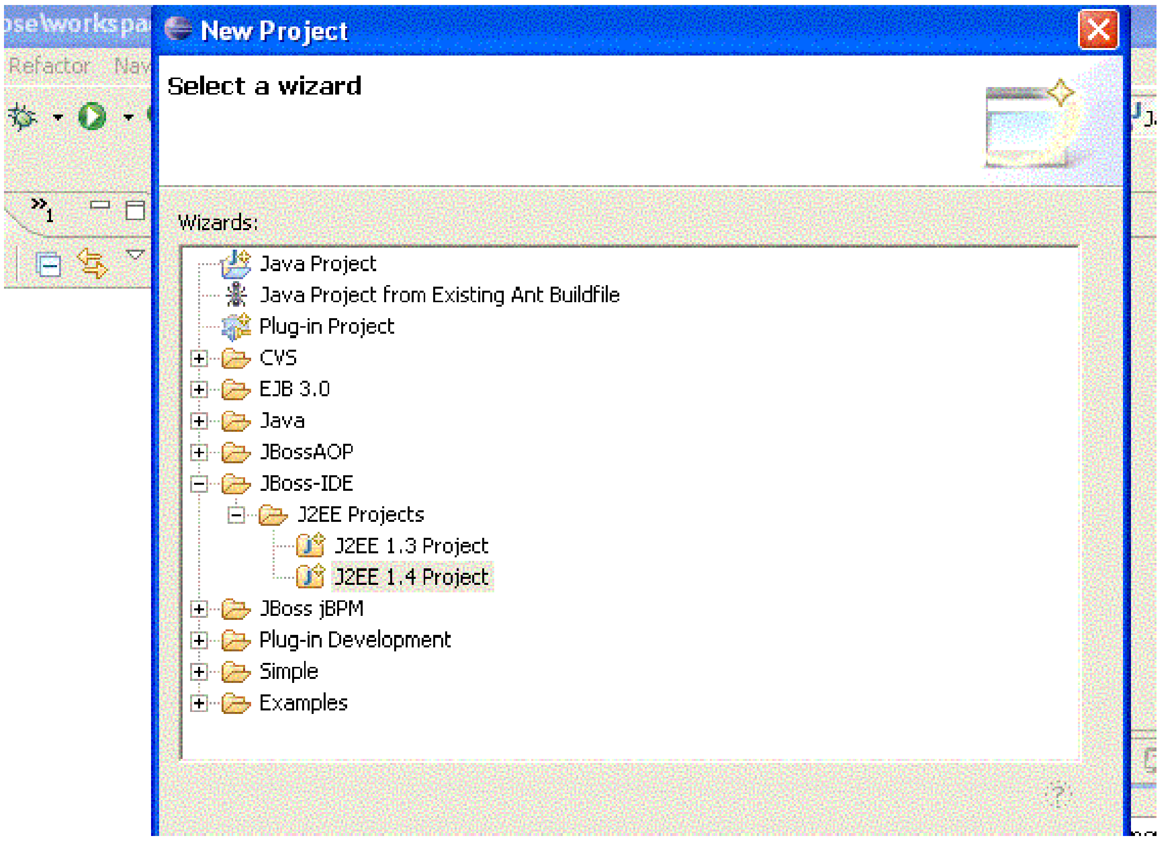
PART 2

Deploying applications

This part provides information about how to deploy various J2EE components at JBoss by using Eclipse. I will not deploy all possible components but only the most fundamental. These components are: JSP, Java Beans, EJB (CMP, Stateful, Stateless), Servlets. I will also provide example how to use security features of J2EE (JAAS) and JavaMail API. At the end I will provide some example of **Ant** deployment (without Eclipse) and **XDoclets** which improves development and makes it easier**.** The aim of this tutorial is not to show how to develop these components, only descriptors and JBoss specific features are interesting. That's why it will not be a lot of Java code. However the code is available for download at zipped format.

Deploying JSP and Java Beans

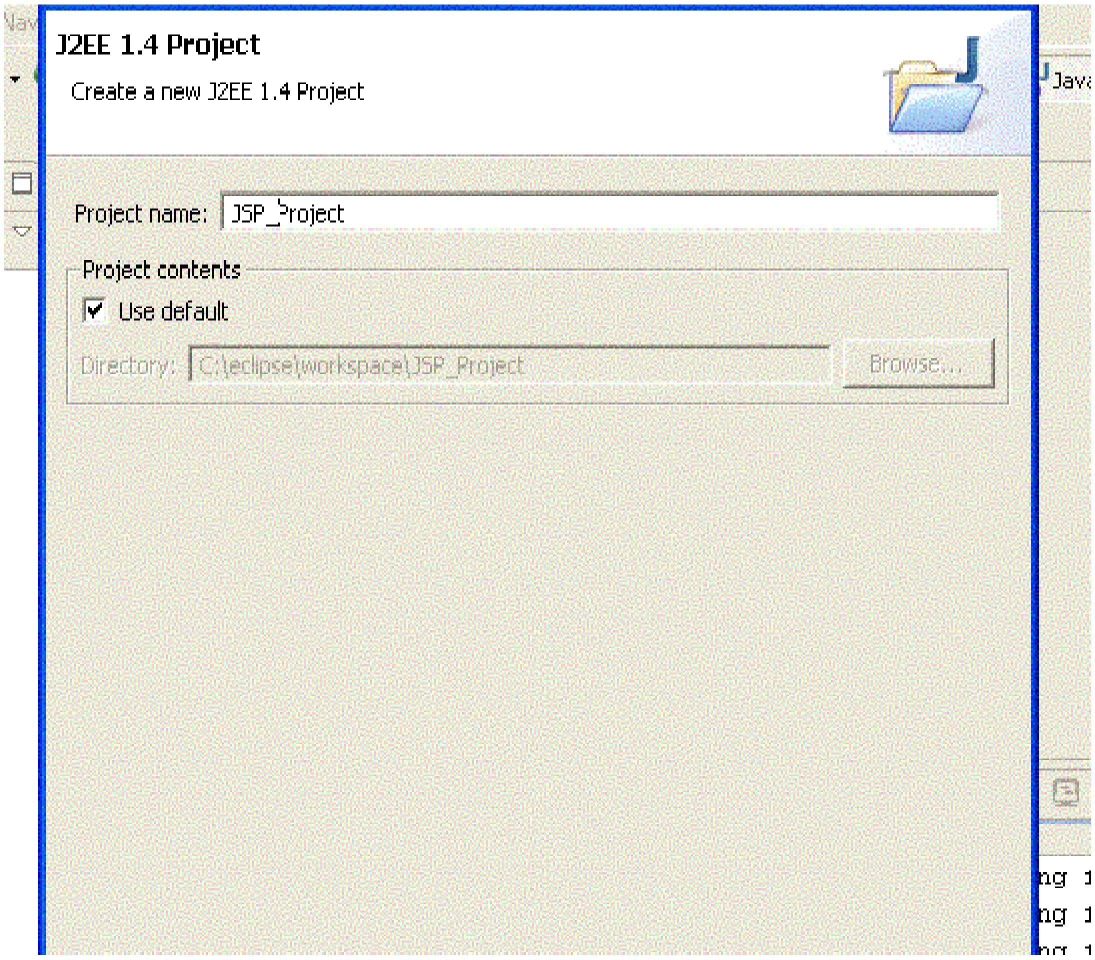
Probably the easiest components to deploy at JBoss are Java Server Pages (JSP) and Java Beans. It is important to understand at very beginning that JBOSS requires certain foldes and certain structure of these folders. If that structure is not correct or if folders have wrong names, nothing will work and deployment will fail! These folders are: src (used for servlets , Java Beans, EJB and all *.java type of code), WEB (JSP,HTML files),WEB-INF (servlet descriptors and some additional libraries), META-INF (ejb descriptors). The structure of the folders will be visible during the demonstrations. JSP and Java Beans are easier to deploy because they don't require descriptors. The next few shots show step-by-step development:



Java Project
Java Project from Existing Ant Buildfile
Plug-in Project
CVS
EJB 3.0
Java
JBossAOP
JBoss-IDE
J2EE Projects
J2EE 1.3 Project
J2EE 1.4 Project
JBoss jBPM
Plug-in Development
Simple
Examples

Figure 14: Creating the J2EE project

.



J2EE 1.4 Project

Create a new J2EE 1.4 Project

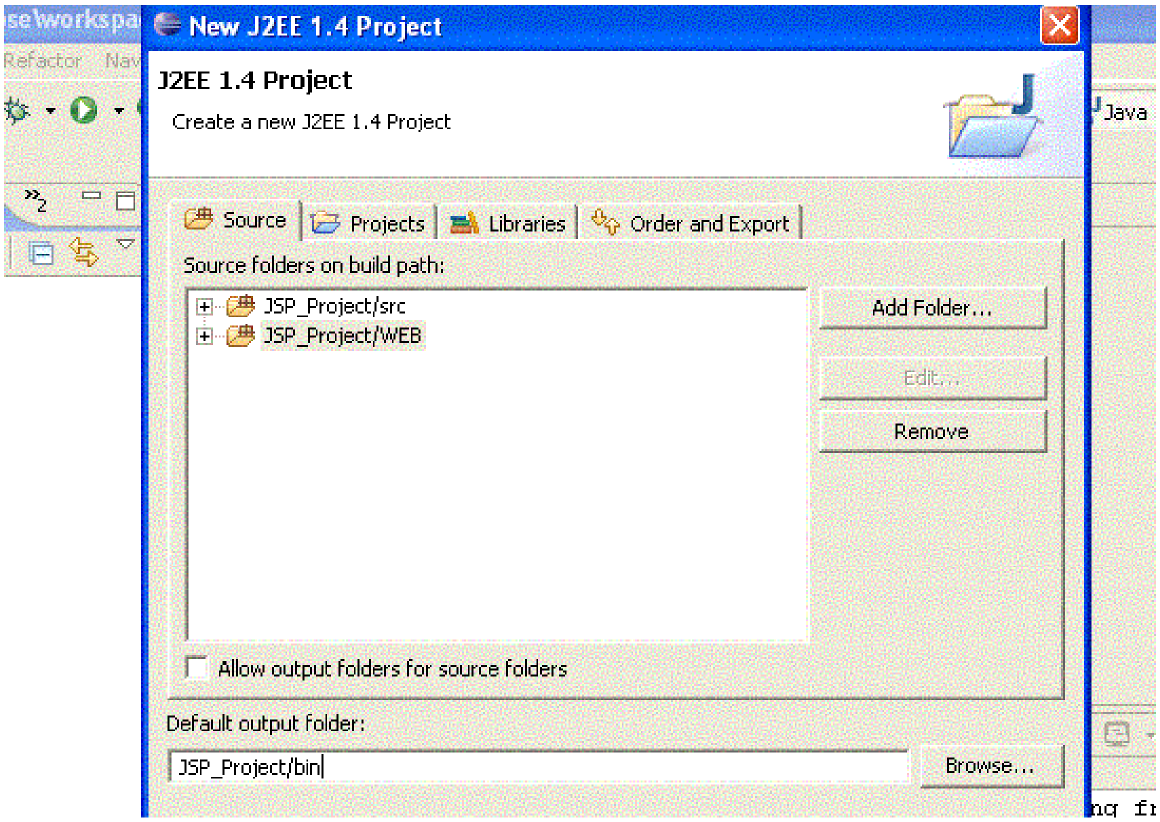
Project name: JSP_Project

Project contents

☑ Use default

Directory: C:\eclipse\workspace\JSP_Project

Figure 15: Project name



J2EE 1.4 Project

Create a new J2EE 1.4 Project

Source folders on build path:

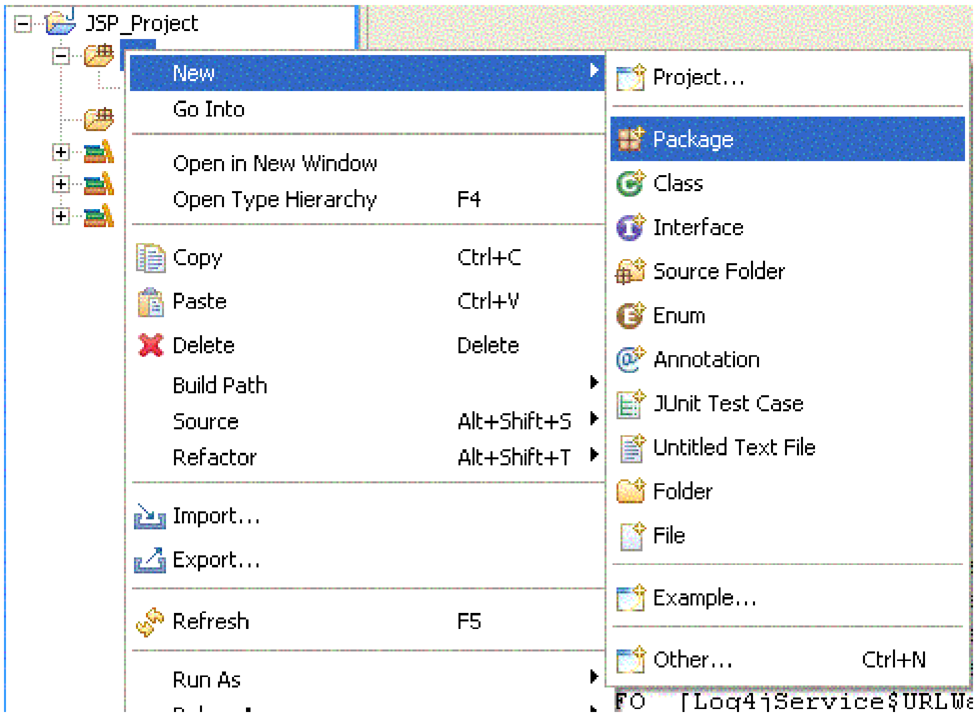
JSP_Project/src
JSP_Project/WEB

Allow output folders for source folders

Default output folder:

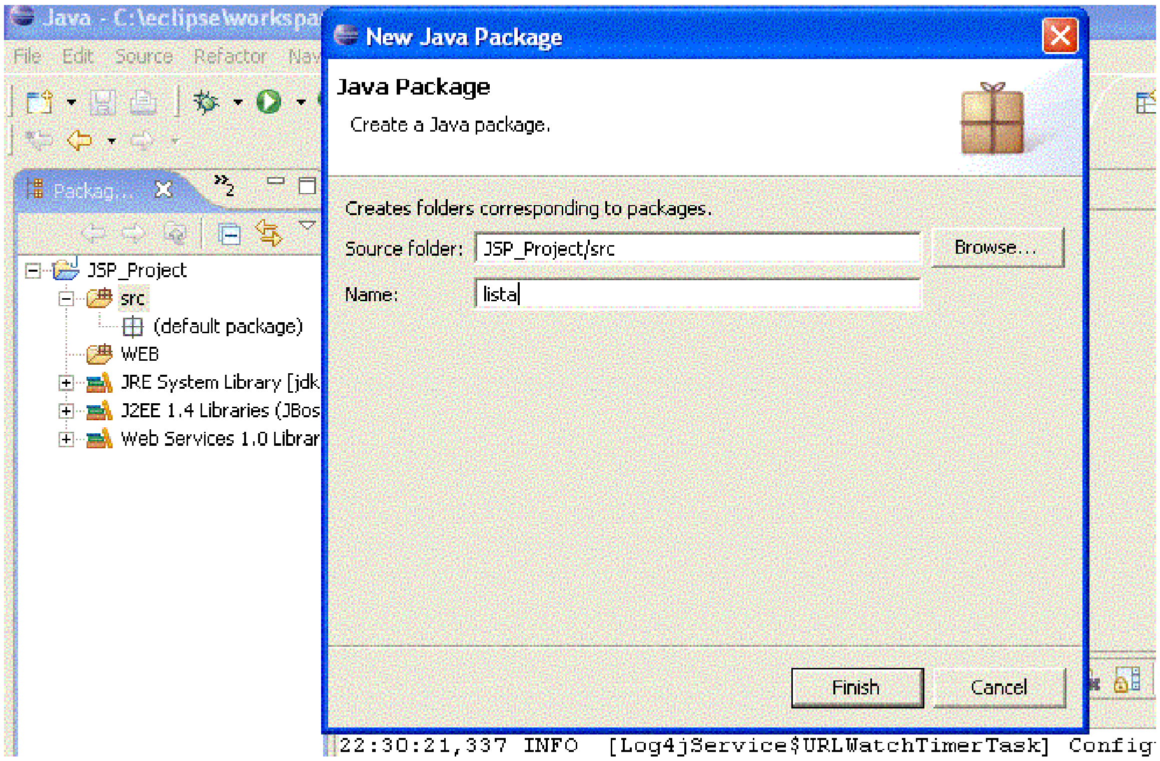
JSP_Project/bin

Figure 16: Add Folder (src,WEB) and default output JSP_Project/bin



```
FO  [Log4jService$URLWa
```

Figure 17: Create new package for Java Bean



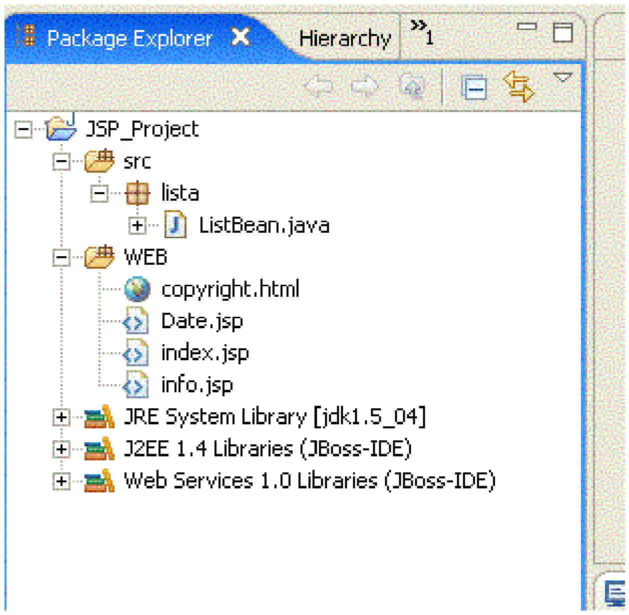
Creates folders corresponding to packages.

Source folder: JSP_Project/src

Name: lista

```
22:30:21,337 INFO  [Log4jService$URLWatchTimerTask] Config
```

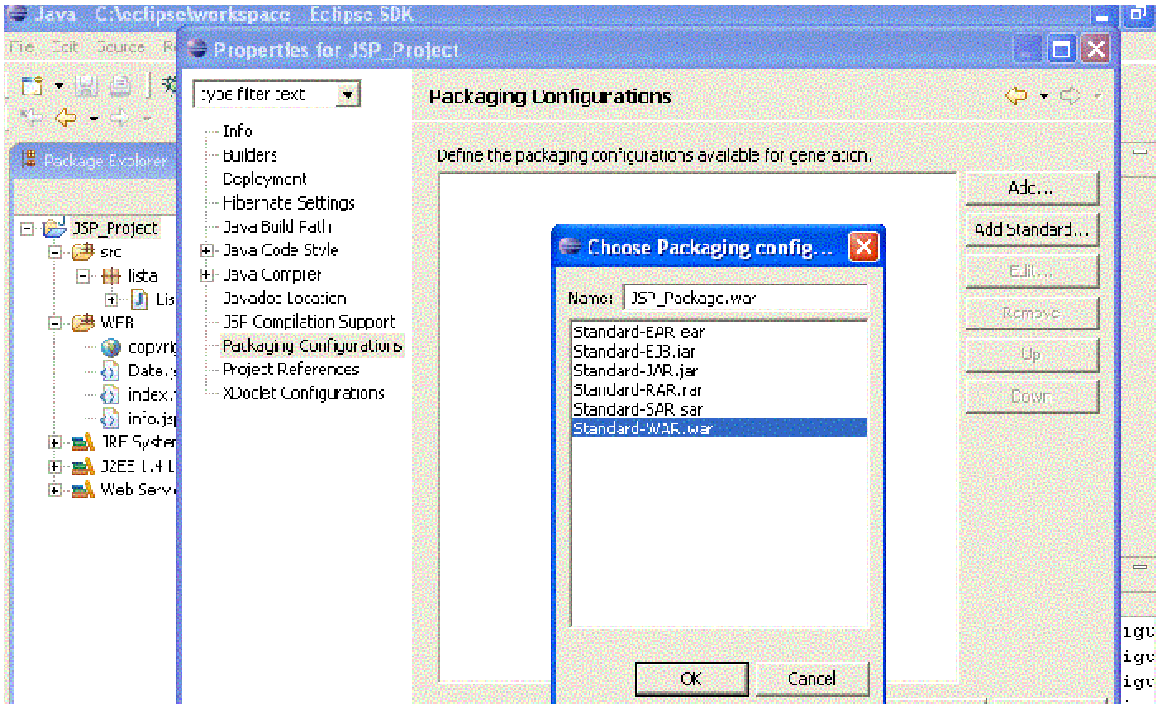
Figure 18: Input the name of the package



```
JSP_Project
  src
    lista
      ListBean.java
  WEB
    copyright.html
    Date.jsp
    index.jsp
    info.jsp
  JRE System Library [jdk1.5_04]
  J2EE 1.4 Libraries (JBoss-IDE)
  Web Services 1.0 Libraries (JBoss-IDE)
```

Figure 19: The structure of application

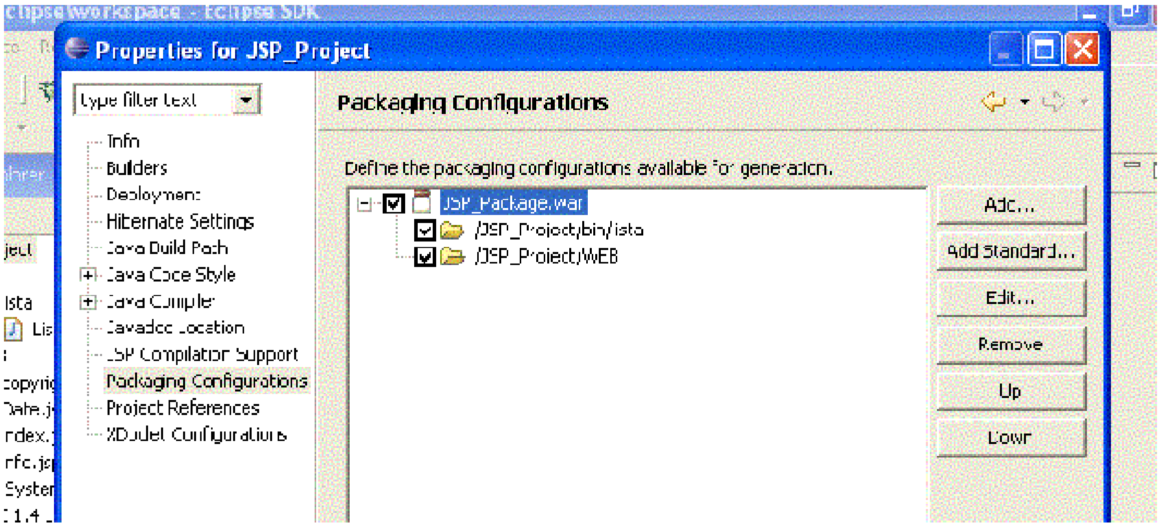
The next step is to compile and package application by using built in facilities (Ant) of Eclipse. The next several shots show how to package appliaction.



Name: JSP_Package.war

Standard-EAR.ear
Standard-EJB.jar
Standard-JAR.jar
Standard-RAR.rar
Standard-SAR.sar
Standard-WAR.war

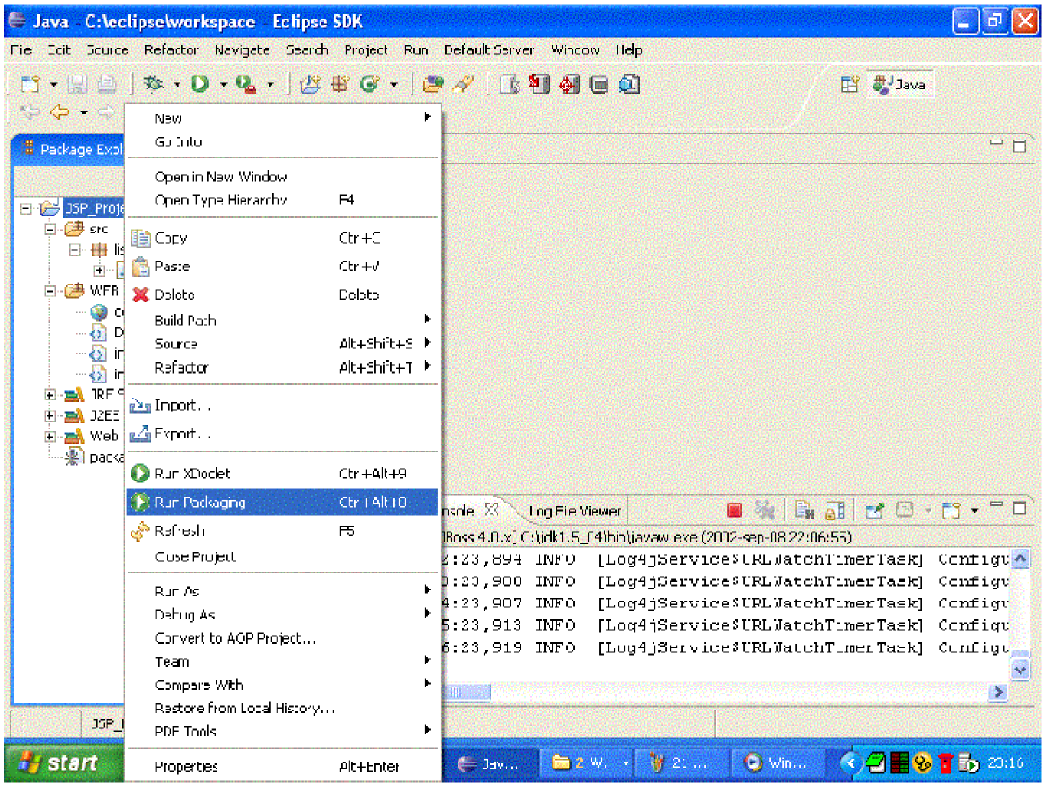
Figure 20: Create a package called JSP_Package.war



JSP_Package.war
/JSP_Project/bin/lista
/JSP_Project/WEB

Figure 21: Right click at package and choose folders (WEB and src)

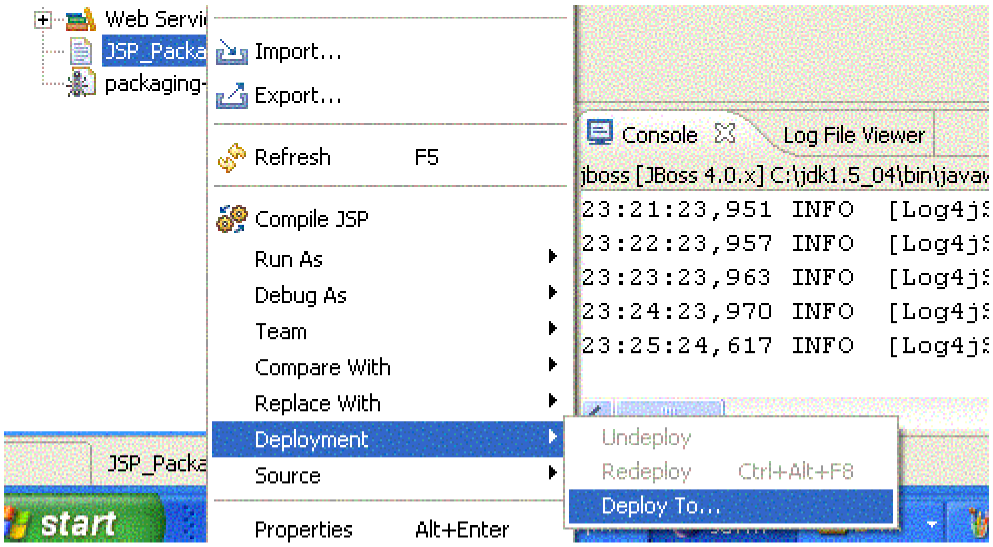
These operations will create **packaging-build.xml file.** By right clicking on project and choosing "Run Packaging" **JSP_Project.war** file will be created.



```
JBoss 4.0.x] C:\jdk1.5_04\bin\javaw.exe (2002-sep-08 22:06:55)
2:23,894 INFO  [Log4jService$URLWatchTimerTask] Configu
0:23,900 INFO  [Log4jService$URLWatchTimerTask] Configu
4:23,907 INFO  [Log4jService$URLWatchTimerTask] Configu
5:23,913 INFO  [Log4jService$URLWatchTimerTask] Configu
6:23,919 INFO  [Log4jService$URLWatchTimerTask] Configu
```

Figure 22: Run packaging

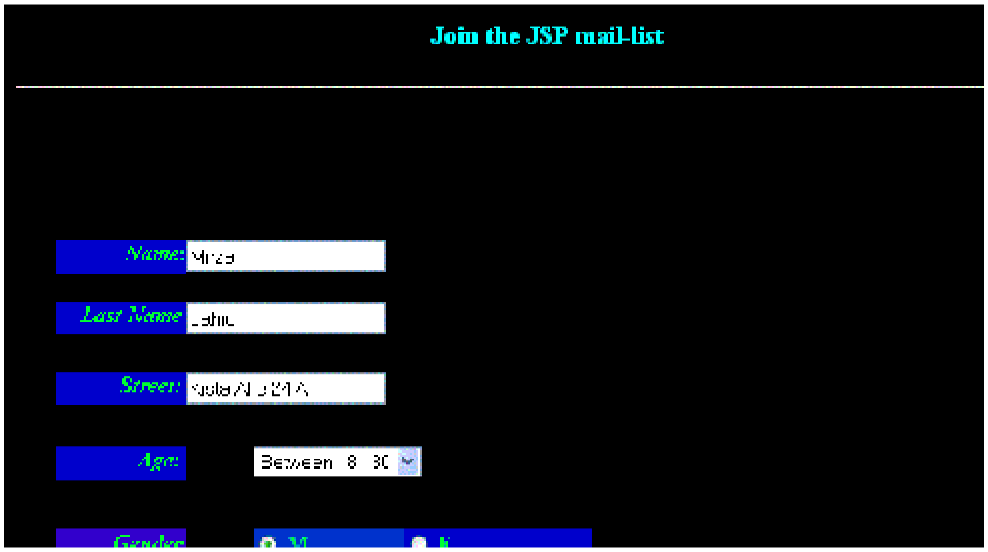
After application has been packaged the only thing left is deployment. Deployment is simple and can be done through the Eclipse or by simply copying the file into deploy folder of JBoss. I will use the first alternative.



```
jboss [JBoss 4.0.x] C:\jdk1.5_04\bin\javaw
23:21:23,951 INFO  [Log4jS
23:22:23,957 INFO  [Log4jS
23:23:23,963 INFO  [Log4jS
23:24:23,970 INFO  [Log4jS
23:25:24,617 INFO  [Log4jS
```

Figure 23: Deploying application at JBoss

After the deployment was done message will be visible at console. Now we can test application by running at: localhost:8081/**JSP_Package/index.jsp.** If everything went OK something the page should be visible.


Join the JSP mail-list

Name:	Mirza
Last Name	[illegible]
Street:	[illegible]
Age:	Between 8 30
Gender	M F

Figure 24: Running JSP and Java Bean

This example is simple but it illustrates how to deploy JSP and Java Beans. If we want to use JSTL tags or any other type of tags we must place them in WEB-INF folder. One more thing to say is that client components (HTML, JSP, Servlets, and Java Beans) must be packed into **.war** type of file, EJB must be packed into .**jar** and both together must be packed into **.ear** files. In order to check if something is missing in the package files can be viewed by using command **jar –tf JSP_Package.war** (for example).

Deploying Servlets and Session EJB

The previous deployment was a soft introduction into more advance deployments. It was also a chance to show some basic work in developing folders and files for an application. Since most of these things are the same I will not repeat the same process all the time but only concentrate on differences.

When deploying Servlets and EJB (in our case 1 stateful and 1 stateless). There are some important things to mention:

- **In order to deploy Servlets into container, you need to create descriptor called web.xml**
- **In order to deploy EJB, you need EJB descriptor called ejb-jar.xml**
- **All EJBs must be packed into single file of *.jar type**
- **All Servlets and other web-components go into file of *.war type**
- **Descriptors go together with components that they describe**
- **You can if you want put together *.jar and *.war file into single file called "Enterprise archive" and has extension *.ear**
- **JBoss server requires also some specific files. In our case jboss.xml (one of them). This file has a purpose to convert JNDI-name into real name. It is not needed if you don't want to use JNDI-name (according to the specification). In my case I used for one bean "ejb/Stateless" and for the other one I didn't (to demonstrate both ways).**

I prefer not to package EJB and web-components together into *.ear file but it is easy to do. The reason is that I can easier make modifications and testing.

After this short introduction I want to say few words about application itself. It is quite simple application that uses a Servlet called "TestServlet.java" and two EJB (Stateful and Stateless). It demonstrates the difference between stateful EJB and stateless EJB. What I mean by that will be visible soon but basic idea is that stateful keeps state and remembers the data while stateless don't. Without going much into details of EJB let see structure of the files and descriptors.

Figure 25 is showing that we have three packages in the **src** folder, **statefulbean** , **statelessbean** and **test** . Each of them contains its components. There are two additional directories, **META-INF** (keeps ejb-jar.xml and jboss.xml descriptors for EJBs), WEB-INF (keeps web.xml descriptors for the TestServlet). So far so good, before seeing those files and exploring them into more details let us see the figure.

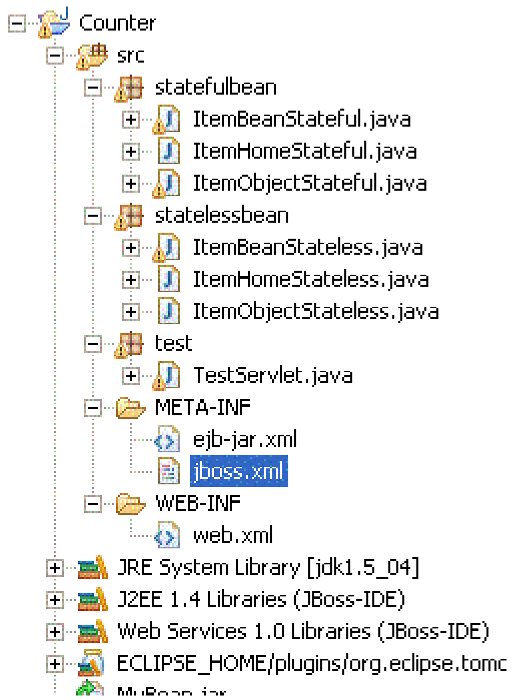


Figure 25: The structure of Counter application

```
<?xml version="1.0" ?>
 <!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN' 'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>

<ejb-jar>
   <enterprise-beans>

      <!-- A minimal session EJB deployment -->
      <session>
         <ejb-name>StatelessBean</ejb-name>
         <home>statelessbean.ItemHomeStateless</home>
         <remote>statelessbean.ItemObjectStateless</remote>
         <ejb-class>statelessbean.ItemBeanStateless</ejb-class>
         <!-- or Stateless -->
         <session-type>Stateless</session-type>
         <transaction-type>Container</transaction-type>
      </session>
      <session>
         <ejb-name>StatefulBean</ejb-name>
         <home>statefulbean.ItemHomeStateful</home>
         <remote>statefulbean.ItemObjectStateful</remote>
         <ejb-class>statefulbean.ItemBeanStateful</ejb-class>
         <!-- or Stateless -->
         <session-type>Stateful</session-type>
         <transaction-type>Container</transaction-type>
      </session>

   </enterprise-beans>
</ejb-jar>
```

This file is called ejb-jar.xml. It declares enterprise-beans in the file and then describes their names, remote and home interfaces and the transaction type.

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>StatelessBean</ejb-name>
      <jndi-name>ejb/Stateless</jndi-name>
    </session>
  </enterprise-beans>
</jboss>
```

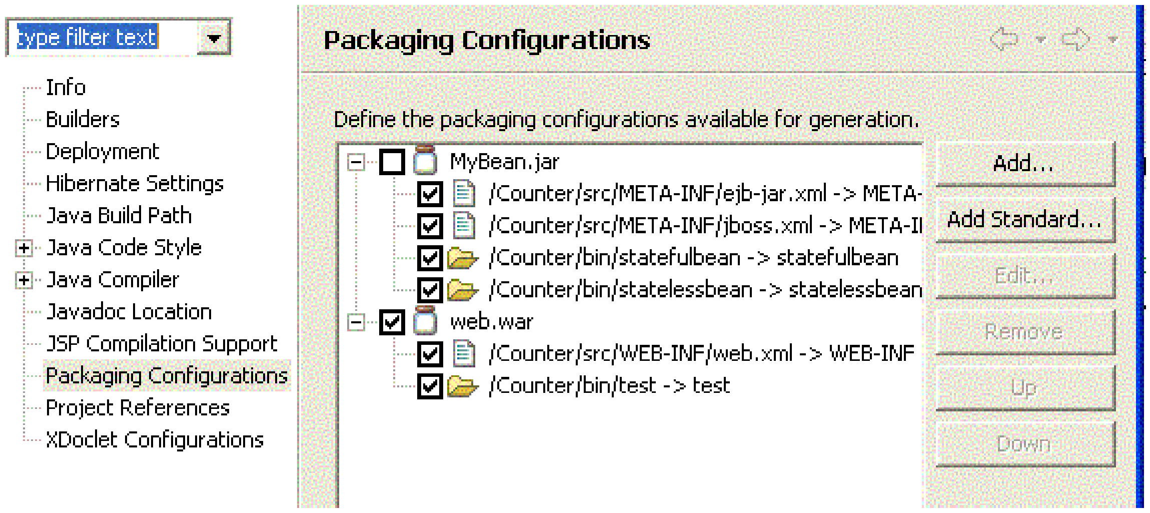
This file is called jboss.xml and it is specific for JBoss server. It helps to the container to locate the EJB by keeping the real name and JNDI-name together.

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee" version="2.4"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>Stateless Items</display-name>


    <servlet>
        <display-name>TestServlet</display-name>
        <servlet-name>TestServlet</servlet-name>
        <servlet-class>test.TestServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>TestServlet</servlet-name>
        <url-pattern>/test</url-pattern>
    </servlet-mapping>
```

```
</web-app>
```

This is the last descriptor and it is used for describing the Servlet called "TestServlet.java". It also gives all important information about this Servlet to the container, like name, classs, url-pattern (context).

Packaging with Eclipse is straight-forward and after right click at Counter project and choosing all needed files the structure of the package should look like the Figure 26.



Define the packaging configurations available for generation.

```
MyBean.jar
    /Counter/src/META-INF/ejb-jar.xml -> META-
    /Counter/src/META-INF/jboss.xml -> META-II
    /Counter/bin/statefulbean -> statefulbean
    /Counter/bin/statelessbean -> statelessbean
web.war
    /Counter/src/WEB-INF/web.xml -> WEB-INF
    /Counter/bin/test -> test
```

Figure 26: Packaging with Eclipse

As I already mentioned, it is possible to put those two files (MyBean.jar and web.war) in the same file. In that case we need one more descriptor called **application.xml**. It should be placed into META-INF directory, together with other files. This is the code of application.xml file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<application xmlns="http://java.sun.com/xml/ns/j2ee" version="1.4"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com /xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/application_1_4.xsd">
<display-name>My Counter</display-name>
<description>Application description</description>
<module>
<ejb>MyBean.jar</ejb>
</module>
<module>
<web>
```

```
<web-uri>web.war</web-uri>
<context-root>/test</context-root>
</web>
</module>
</application>
```

This file simply describes **MyBean.jar** (that contains EJBs) and **web.war (**Servlets) . It also shows the context-root of the application. The name could be any but it must end with *.ear.

Running the application

Running of this small but stil enterprise application is simple. By using URL: http://localhost:8081/web/test should display the page like in Figure 27.

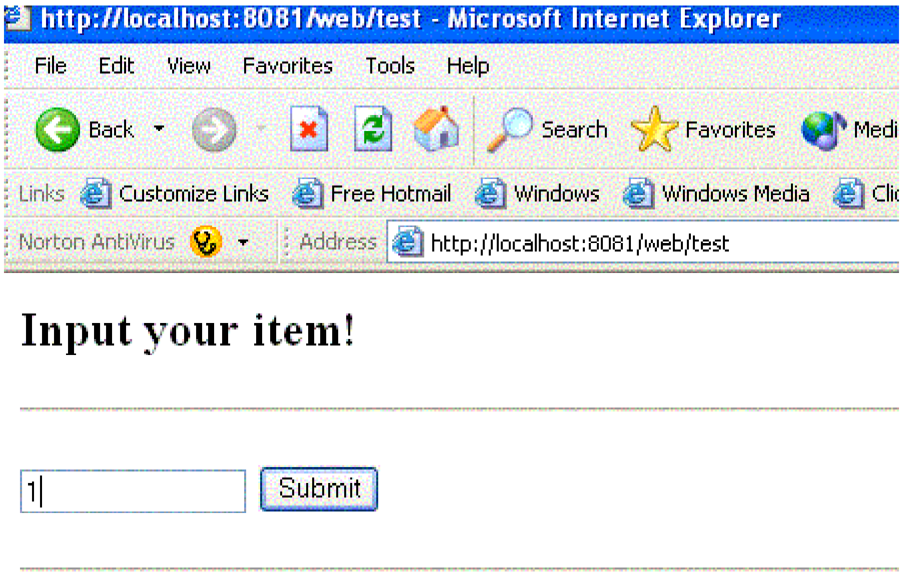
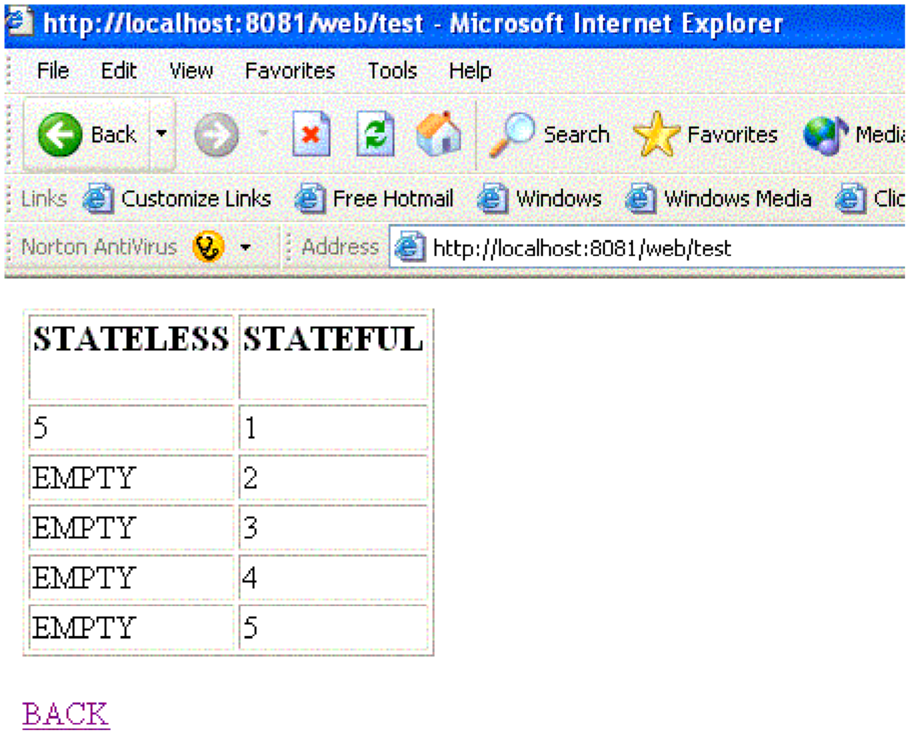

http://localhost:8081/web/test - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Medi
Links Customize Links Free Hotmail Windows Windows Media Cli
Norton AntiVirus Address http://localhost:8081/web/test
Input your item!
1
Submit

Figure 27: Inputting the item (number) into stateful and stateless bean

After several inputs (5) Figure 28 shows that the stateless bean will keep only the last item (in this case 5), the last state and he stateful will keep all items (states).


http://localhost:8081/web/test - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Medi
Links Customize Links Free Hotmail Windows Windows Media Clic
Norton AntiVirus Address http://localhost:8081/web/test

STATELESS	STATEFUL
5	1
EMPTY	2
EMPTY	3
EMPTY	4
EMPTY	5

BACK

Figure 28: Stateful vs. Stateless EJB

This is the end of this example. It was not some big and very exciting example but it showed some fundamental steps in packaging and deploying Session Beans and Servlets. It also showed how to create descriptors for these files.

Deploying Servlets Entity Beans and Session Beans

Next example is more advanced than previous two. The reason is the fact that I'm going to use more components. I'm going to use Servlets, Session Beans (stateless and two stateful) and CMP Entity Bean. Beside that I will introduce one of the most basic EJB patterns called "Session Facade". Before that I will say few words about Entity Beans and their purpose.

Short introduction to Entity Beans

Entity Beans is an EJB component that saves its state into database. That means we can keep data much longer (as a table in a database). There are two types of Entity Beans CMP and BNP Beans. There is also JMS, which is kind of Entity Bean but is used, in different circumstances. BNP means "Bean Managed Persistence" and it is simply managed by the bean itself. Data is saved and removed from the tables **explicitly** with a help of methods made by a developer. CNP means "Container Managed Persistence" and it saves also data into database but this time the job is done by the container, so the data is saved **implicitly**.
User simply passes data to the container and container does the job. This time, the container implements all needed methods not the developer.

Short introduction to EJB Design Patterns

In order to minimize the number of mistakes and to make the development easier and faster many developers use already proven methods/practices. These practices are called "**Design Patterns**". The most widely used EJB-Design Pattern is called "**Session Facade**". The basic idea behind the session facade is simply to create a stateless bean and from that bean control and run all other beans, the most often entity and stateful beans. In this example I used this type of EJB-Design Pattern. Figure 29 shows the structure of this example. More information about in **book "EJB Design Pattern**" by Floyd Marinescu.

Student Application

Student application has a task to save information about students into database. In order to do that it also uses other J2EE components: Servlets (Client and FinalServlet), Stateless bean (Stateless Control) , Stateful beans (StatefulA and StatefulB) , class (EJBFactory) and one CMP bean (StudentBean). Figure 32 shows the structure of the application.

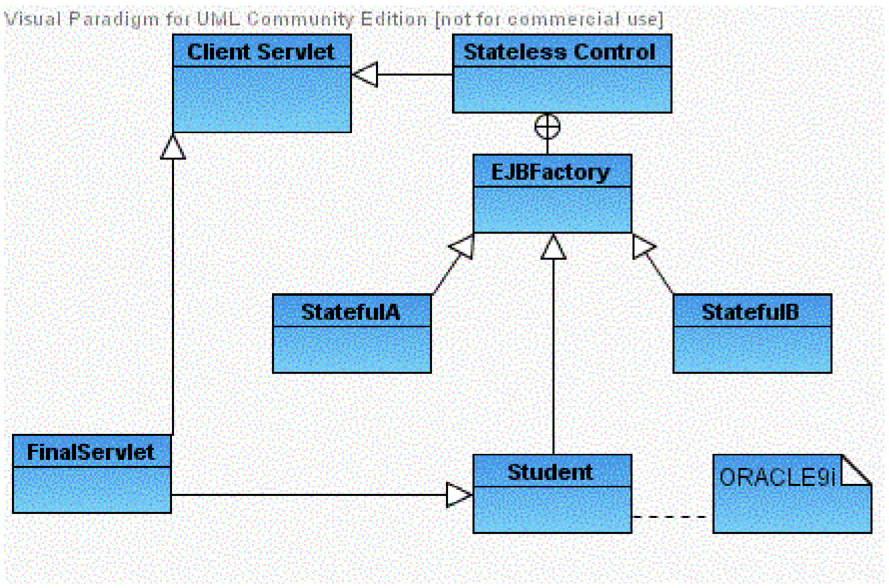


Figure 29: Student application

Packaging of application

Now when everything is clear and all classes and descriptors have been created it's time to package it and deploy it. After that try to run it. Figure 30a and Figure 30b shows all packages and folders that are needed for this application. It also shows the structure of the application.

I generated two packages, **MyBeans.jar** (keeps EJBs) and **webclient.war** (keeps Servlets). Before clicking the right button and running packaging, the packages (MyBeans.jar and webclient.war) must be packaged like in the figure 31.

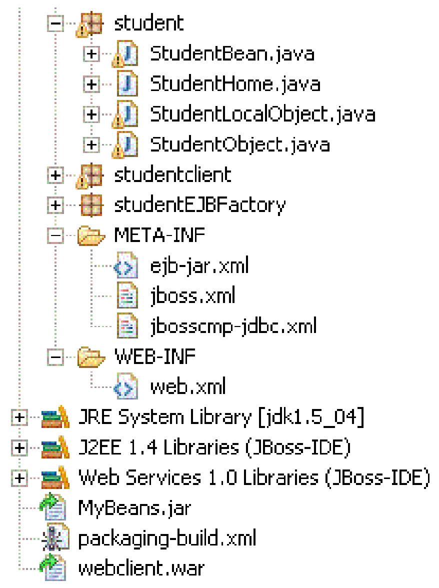


Figure 31: Structure of packages

When packages have been created (right click on Student ->“Run Packaging”) and the two packages have been generated it remains only to deploy them. Again, it can be done through the Eclipse function “Deploy” (right click) or by simply copying the files into “deploy” folder of JBoss.

Descriptors

This application uses several file descriptors. This file is called **ejb-jar.xml** and just like in the previous example it is used by the EJBs. The only difference is that we have one entity bean called “Student”. Entity beans are specific and we see it from the descriptor.

For example this segment of code:

```
<cmp-field>
   <field-name>id</field-name>
</cmp-field>
```

This segment describes the cmp-field called id. The filed will have same name in the constructor. So simply said cmp-fields map to the table in a database. The next interesting parts of the cmp-bean are tags that describe EJB-queries.
Something like this :

```
<query-method>
   <method-name>findAll</method-name>
<method-params>

<ejb-ql>SELECT DISTINCT OBJECT(p) FROM StudentBean p WHERE p.id IS NOT
NULL
</ejb-ql>
```

Both fragments of this code work together. The first one defines method **findAll**, used to query the database through the bean and the second simply defines that SQL-look-a-like query (EJB-QL). There is one more query there called “findByPrimaryKey”. Here is id used as a unique primary constraint. This method is simply implemented by the container and doesn’t need any EJB-QL. Apart from that there is nothing else worth mentioning.

```
<?xml version="1.0" ?>
 <!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN' 'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>

<ejb-jar>
      <enterprise-beans>


           <session>

              <ejb-name>StatefulB</ejb-name>
                <home>statefulEJBB.StatefulBHome</home>
                  <remote>statefulEJBB.StatefulBObject</remote>
                  <ejb-class>statefulEJBB.StatefulB</ejb-class>
                  <session-type>Stateful</session-type>
            <transaction-type>Container</transaction-type>

         </session>
         <session>

              <ejb-name>StatelessFacade</ejb-name>
                <home>statelessEJBPattern.StatelessControlHome</home>

      <remote>statelessEJBPattern.StatelessControlObject</remote>
                  <ejb-class>statelessEJBPattern.StatelessControl</ejb-
class>
                  <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>

             </session>
            <session>

              <ejb-name>StatefulA</ejb-name>
                <home>statefulEJBA.StatefulAHome</home>
                  <remote>statefulEJBA.StatefulAObject</remote>
                  <ejb-class>statefulEJBA.StatefulA</ejb-class>
                  <session-type>Stateful</session-type>
            <transaction-type>Container</transaction-type>

        </session>
        <entity>
                  <ejb-name>Student</ejb-name>

                  <home>student.StudentHome</home>
                  <remote>student.StudentObject</remote>
                  <ejb-class>student.StudentBean</ejb-class>
                  <persistence-type>Container</persistence-type>
                  <prim-key-class>java.lang.String</prim-key-class>
```

```
                         <reentrant>false</reentrant>

                    <cmp-version>2.x</cmp-version>
                   <abstract-schema-name>StudentBean</abstract-schema-name>
                   <cmp-field>
                       <field-name>id</field-name>
                   </cmp-field>
                   <cmp-field>
                       <field-name>name</field-name>
                   </cmp-field>
                   <cmp-field>
                       <field-name>lastName</field-name>
                   </cmp-field>
                   <cmp-field>
                       <field-name>year</field-name>
                   </cmp-field>
                   <cmp-field>
                       <field-name>uni</field-name>
                   </cmp-field>
                   <cmp-field>
                       <field-name>subject</field-name>
                   </cmp-field>
                         <primkey-field>id</primkey-field>
                         <query>
                       <query-method>
                           <method-name>findAll</method-name>
                           <method-params>

                           </method-params>
                       </query-method>
                       <ejb-ql>SELECT DISTINCT OBJECT(p) FROM StudentBean p
WHERE p.id IS NOT NULL</ejb-ql>
                   </query>

                   </entity>

             </enterprise-beans>
</ejb-jar>
```

The next file that comes with Student project is called **jboss-cmp-jdbc.xml**.

```
<!DOCTYPE jbosscmp-jdbc PUBLIC
          "-//JBoss//DTD JBOSSCMP-JDBC 4.0//EN"
          "http://www.jboss.org/j2ee/dtd/jbosscmp-jdbc_4_0.dtd">
<jbosscmp-jdbc>
    <defaults>
      <datasource>java:/XAOracleDS</datasource>
      <datasource-mapping>Oracle9i</datasource-mapping>
       <create-table>true</create-table>
        <alter-table>false</alter-table>
        <remove-table>false</remove-table>
    </defaults>
    <enterprise-beans>
        <entity>
            <ejb-name>Student</ejb-name>
            <table-name>Grads</table-name>
            <cmp-field>
                <field-name>id</field-name>
                <column-name>Studentid</column-name>
            </cmp-field>
            <cmp-field>
                <field-name>name</field-name>
                <column-name>Name</column-name>
            </cmp-field>
            <cmp-field>
                <field-name>lastName</field-name>
                <column-name>LastName</column-name>
            </cmp-field>
             <cmp-field>
                <field-name>year</field-name>
                <column-name>Year</column-name>
            </cmp-field>
            <cmp-field>
                <field-name>uni</field-name>
                <column-name>Uni</column-name>
            </cmp-field>
            <cmp-field>
                <field-name>subject</field-name>
                <column-name>Course</column-name>
            </cmp-field>

        </entity>
     </enterprise-beans>
</jbosscmp-jdbc>
```

This file is CMP specific. It is used only with CMP beans. This fragment of code is probably the most interesting. It is very important to declare the DataSource of your database where the data will be saved. In my case ORACLE DataSource. If you are using Hypersonic database you don't need to do that. If you want to create table during the runtime set it to true, you probably don't want to modified tables so set all to false.

```
<defaults>
      <datasource>java:/XAOracleDS</datasource>
      <datasource-mapping>Oracle9i</datasource-mapping>
       <create-table>true</create-table>
        <alter-table>false</alter-table>
        <remove-table>false</remove-table>
</defaults>
```

The next fragment of code defines the name of the bean and the name of the table where everything will be saved.

```
<ejb-name>Student</ejb-name>
   <table-name>Grads</table-name>
<cmp-field>
```

The rest of the file is just mapping between CMP-fields and tables in the database.

All other files have already been commented in the previous project and I will just list them.

This is **jboss.xml** file. It is deployed in META-INF directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss>


  <enterprise-beans>
    <session>
       <ejb-name>StatefulB</ejb-name>
       <jndi-name>StatefulBeanB</jndi-name>
    </session>
    <session>
       <ejb-name>StatefulA</ejb-name>
       <jndi-name>StatefulBeanA</jndi-name>
    </session>
    <session>
       <ejb-name>StatelessFacade</ejb-name>
       <jndi-name>SessionFacade</jndi-name>
    </session>
  </enterprise-beans>enterprise-beans>

</jboss>
```

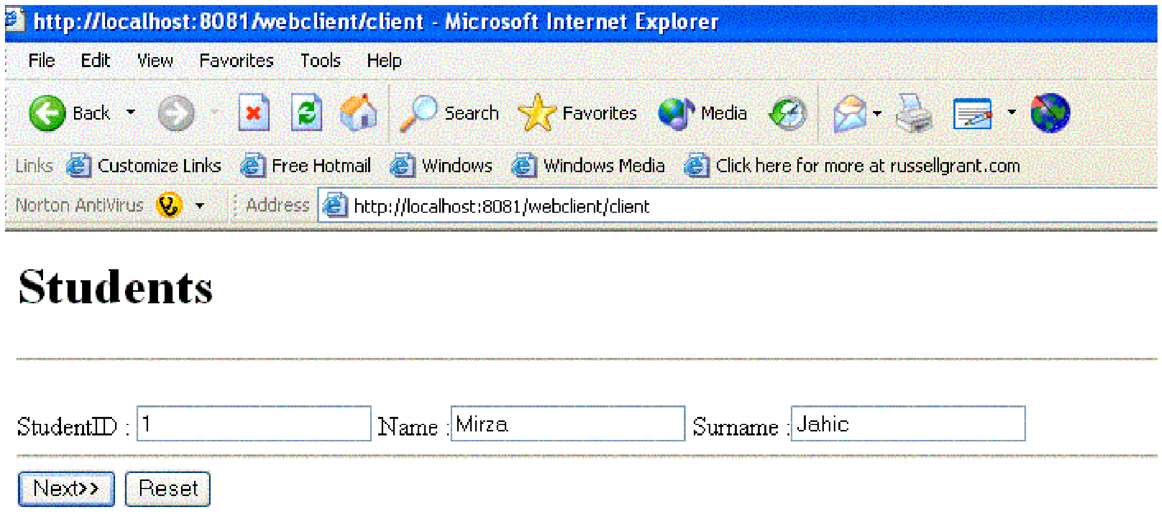
The last file is web.xml and it is deployed in WEB-INF. As already mentioned, it is used to describe Servlets.

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee" version="2.4"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

      <servlet>
            <servlet-name>Client</servlet-name>
            <servlet-class>studentclient.Client</servlet-class>
      </servlet>
      <servlet>
            <servlet-name>FinalServlet</servlet-name>
            <servlet-class>studentclient.FinalServlet</servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>Client</servlet-name>
            <url-pattern>/client</url-pattern>
      </servlet-mapping>
      <servlet-mapping>
            <servlet-name>FinalServlet</servlet-name>
            <url-pattern>/final</url-pattern>
```

```
    </servlet-mapping>
</web-app>
```

Running the application

Well now everything has been properly packaged and deployed it's time to test application. This application queries by using 2 find-methods. User can choose which one to use.



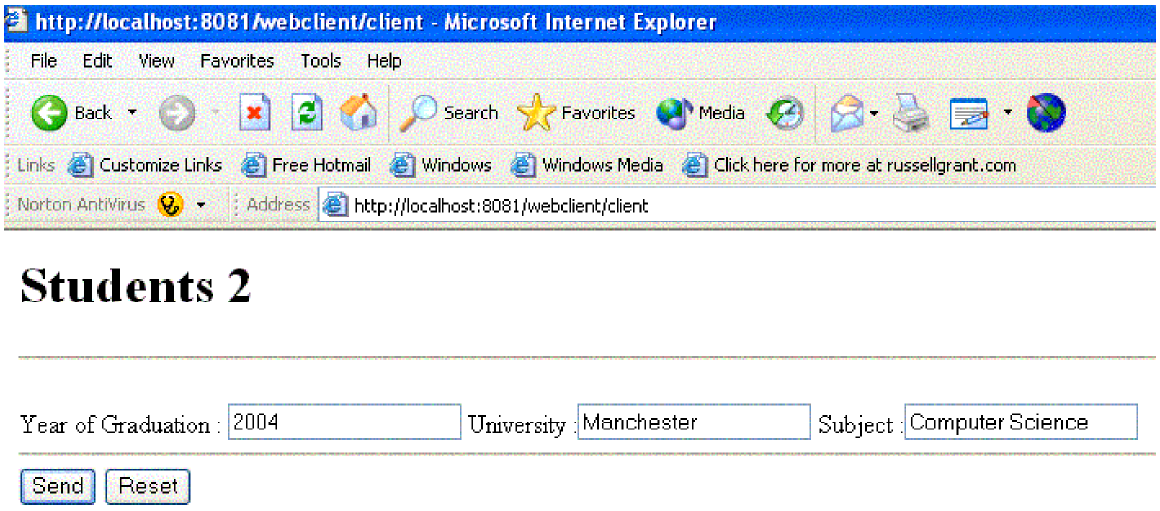
http://localhost:8081/webclient/client - Microsoft Internet Explorer

Address: http://localhost:8081/webclient/client

Students

StudentID : 1 Name : Mirza Surname : Jahic

Figure 32: Fill out the first form

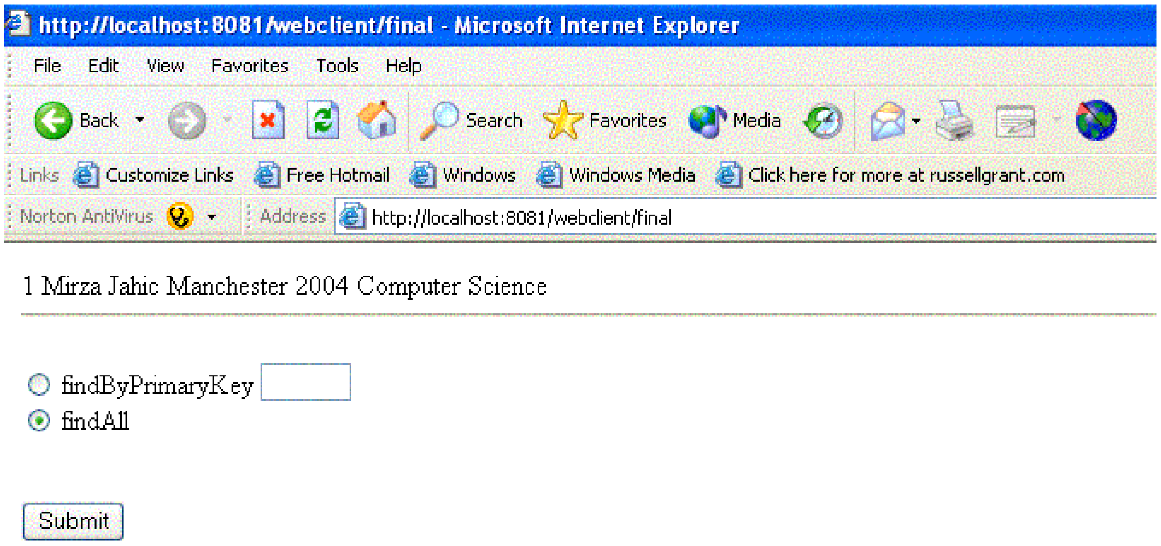

http://localhost:8081/webclient/client - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Customize Links Free Hotmail Windows Windows Media Click here for more at russellgrant.com
Norton AntiVirus Address http://localhost:8081/webclient/client

Students 2

Year of Graduation : 2004 University : Manchester Subject : Computer Science

Send Reset

Figure 33: Fill out the form 2

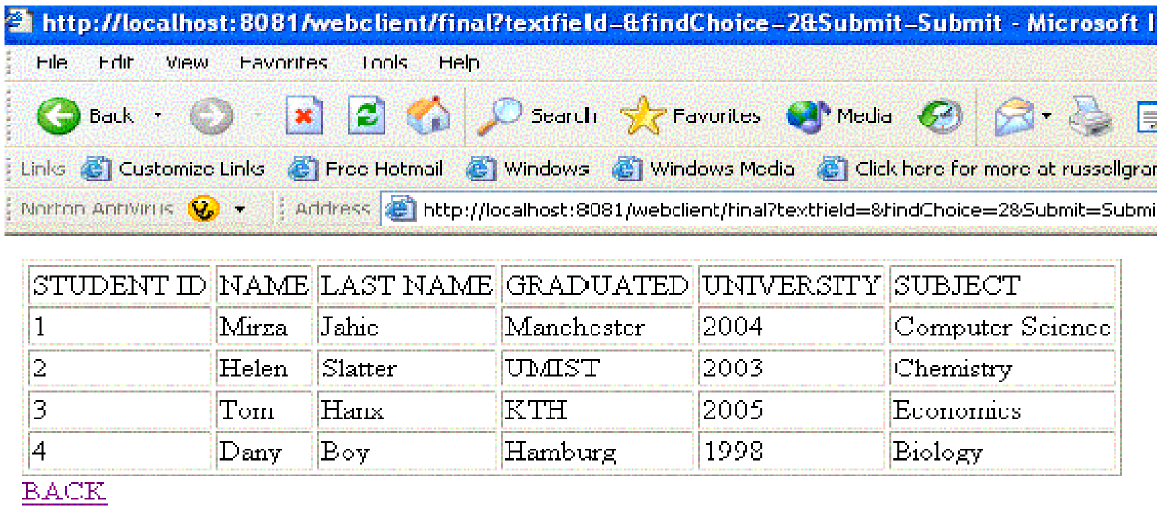

http://localhost:8081/webclient/final - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Customize Links Free Hotmail Windows Windows Media Click here for more at russellgrant.com
Norton AntiVirus Address http://localhost:8081/webclient/final

1 Mirza Jahic Manchester 2004 Computer Science

○ findByPrimaryKey
[✓ findAll]

Submit

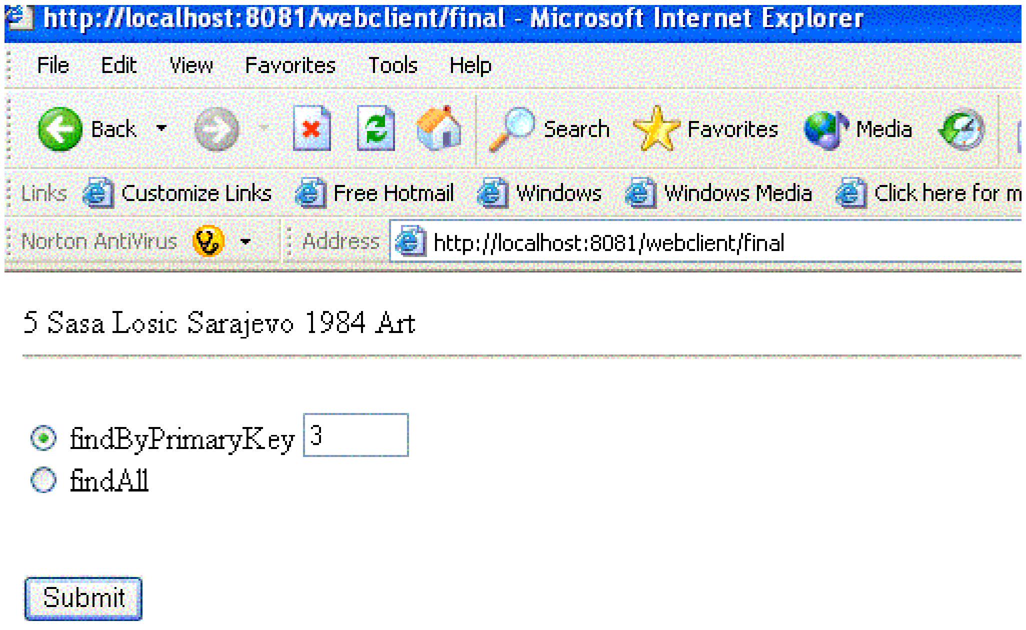
Figure 34: Find all in database


http://localhost:8081/webclient/final?textfield=&findChoice=2&Submit=Submit - Microsoft I
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Links
Customize Links
Free Hotmail
Windows
Windows Media
Click here for more at russellgrar
Norton AntiVirus
Address
http://localhost:8081/webclient/final?textfield=&findChoice=2&Submit=Submi
BACK

STUDENT ID	NAME	LAST NAME	GRADUATED	UNIVERSITY	SUBJECT
1	Mirza	Jahic	Manchester	2004	Computer Science
2	Helen	Slatter	UMIST	2003	Chemistry
3	Tom	Hanx	KTH	2005	Economics
4	Dany	Boy	Hamburg	1998	Biology

Figure 35: Retrieve all from the database

If we choose next time to use primary key and to retrieve only 1 record, it will look like this.

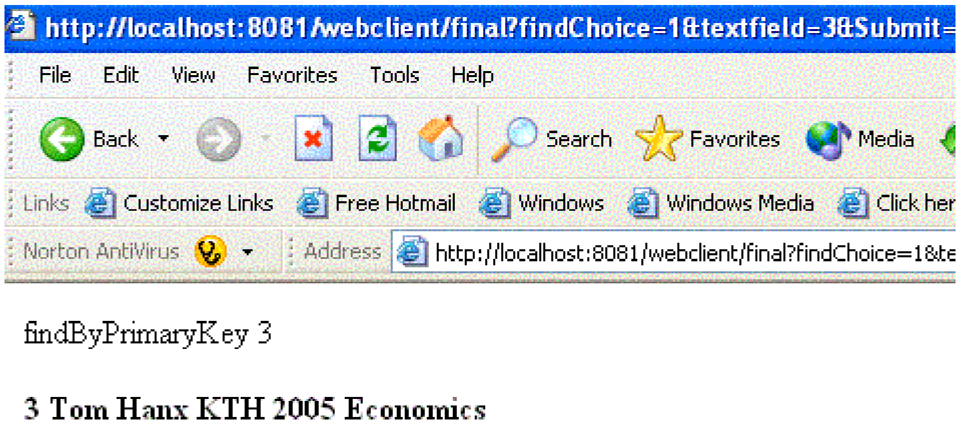

http://localhost:8081/webclient/final - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Customize Links Free Hotmail Windows Windows Media Click here for m
Norton AntiVirus Address http://localhost:8081/webclient/final

5 Sasa Losic Sarajevo 1984 Art

(● findByPrimaryKey) 3
○ findAll

Submit

Figure 36: Get record 3 from the database


http://localhost:8081/webclient/final?findChoice=1&textfield=3&Submit=
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Customize Links Free Hotmail Windows Windows Media Click her
Norton AntiVirus Address http://localhost:8081/webclient/final?findChoice=1&te

findByPrimaryKey 3

3 Tom Hanx KTH 2005 Economics

Figure 37: Record number 3 retrieved

Well, that is basically all when it comes to using CMP beans. Of course this is just an introduction and not all features have been covered, but this gives quite good idea how to deploy this kind of beans by using Eclipse and JBoss. In the next section we are going to cover security (basic) and JavaMail API with JBoss.

Security issues and JavaMail API

This section is the last of the Part2. I want to say few words about security issues in JBoss and then to finish with a simple JavaMail API implementation.

JBoss provides several types of security. The most basic is "**security_check**" type of security but there are also possibilities to provide more advanced features like "**Custom Security**" that uses components called filters in its implementation. JBOSS-LDAP security offers more robust features. Lightweight Directory Protocol provides connection that can be used to store information into LDAP-repository.

Introduction to JAAS

So, JAAS is Java Authentication and Authorization Service and it is a standard for all J2EE application. JBoss also uses this standard. The Java Authentication and Authorization Service (JAAS) is a set of APIs that can be used for two purposes:

- For authentication of users, to reliably and securely determine who is currently executing Java code, regardless of whether the code is running as an application, an applet, a bean, or a servlet; and
- For authorization of users, to determine reliably and securely who is currently executing Java code, regardless of whether the code is running as an application, an applet, a bean, or a servlet; and

JAAS authentication is performed in a pluggable fashion. This permits Java applications to remain independent from underlying authentication technologies. New or updated technologies can be plugged in without requiring modifications to the application itself. JAAS authorization extends the existing Java security architecture that uses a security policy to specify what access rights are granted to executing code (*Sun*).

Introduction to JavaMail API

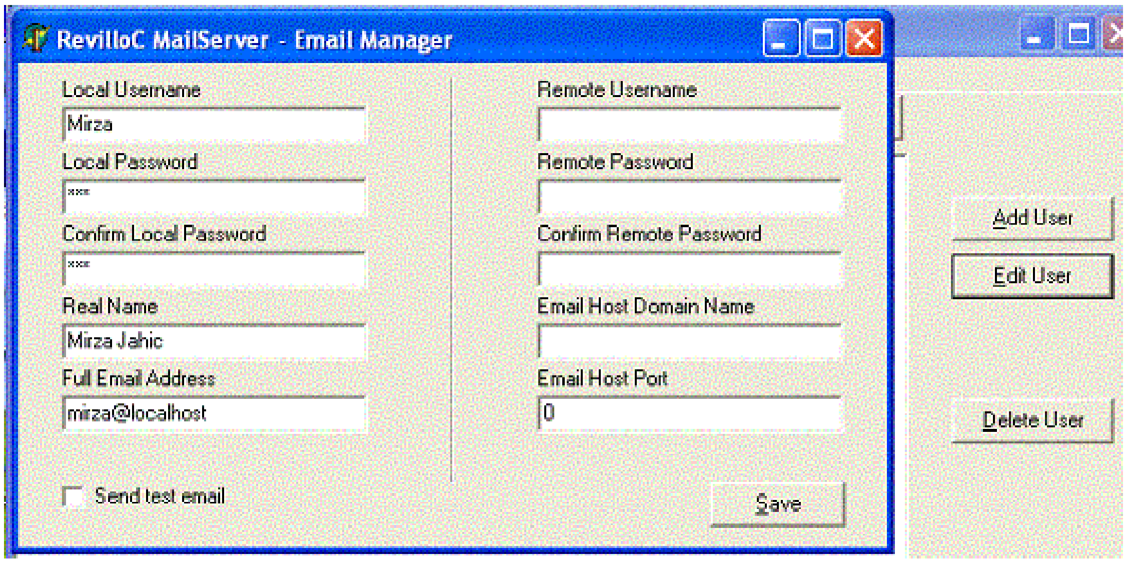
The JavaMail API provides a platform-independent and protocol-independent framework to build mail and messaging applications. The JavaMail API is implemented as a Java platform optional package and is also available as part of the Java platform, Enterprise Edition. JavaMail uses three the most popular protocols: SMTP (for sending emails), POP3 (for receiving emails) and IMAP (more advance than POP3).

Sending authorized messages using Java Mail API and JAAS

This simple application demonstrates how to use JAAS and JavaMail in the same time. The simple idea is to create an application that will authorize a user by using JAAS and if positive allow him to send a message to a local mail server. The message will be end accessed by using some mail client like “Outlook Express”. Information about the users will be kept in local database in two tables (users and userroles). Application will use two JSP, one servlet and one session bean. The code of JavaMail for sending will be in the bean, in a method called “send.

Setting up the mail server

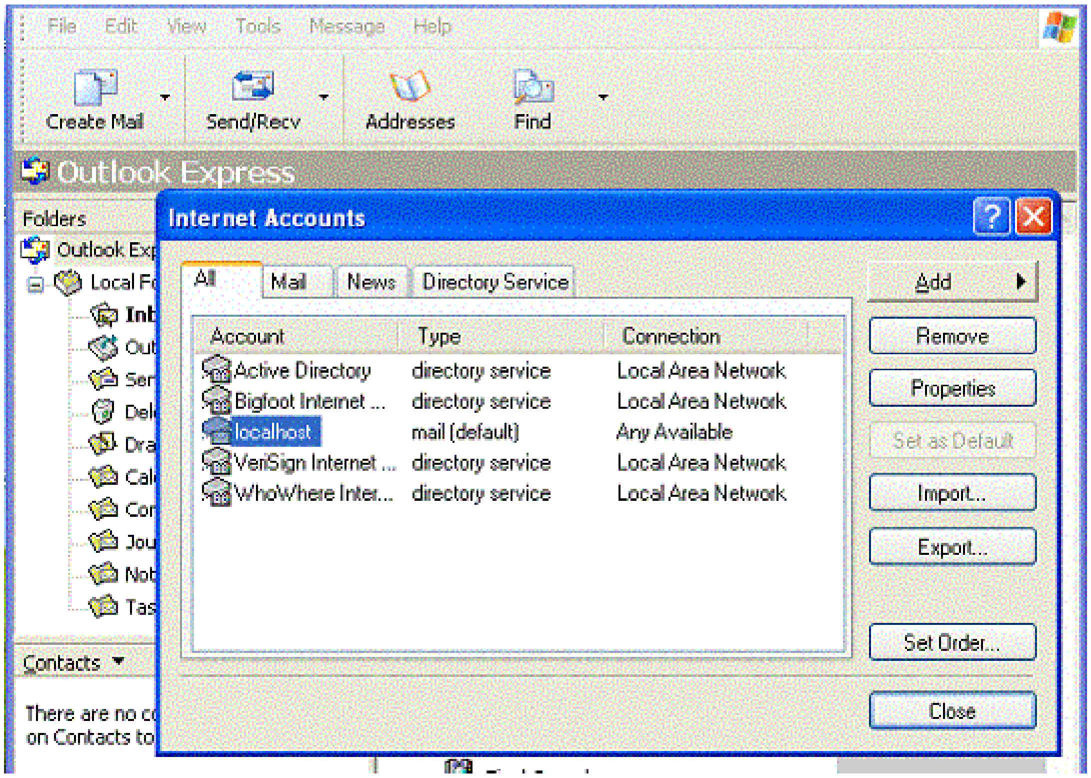
Before any deployment and packaging can be done, the mail server and the mail client must be setup. In my case I will simply use **localhost** and everything will be at one machine. When it comes to mail server I've chosen **RevilloC,** free to download, easy to use, looks nice. After installing RevilloC I needed to create a user and its email. To access that particular account (from the client) I've created username and the password. **Figures 39** shows how the setup should look like.



Local Username: Mirza
Local Password: ***
Confirm Local Password: ***
Real Name: Mirza Jahic
Full Email Address: mirza@localhost
Remote Username:
Remote Password:
Confirm Remote Password:
Email Host Domain Name:
Email Host Port: 0

Figure 39: Creating the user in RevilloC mail server

In order to be able to retrieve sent message from the server you need to configure mail client. In my case, I use "Express Outlook". Simply choose **Tools->Accounts** and then add new account. The **Figure 40** shows the details.



Account	Type	Connection
Active Directory	directory service	Local Area Network
Bigfoot Internet ...	directory service	Local Area Network
localhost	mail (default)	Any Available
VeriSign Internet ...	directory service	Local Area Network
WhoWhere Inter...	directory service	Local Area Network

Figure 40: Creating the new mail account on Outlook

These configurations should be enough for this application, to send and retrieve the mail.

Deploying the application

The most difficult part of this application is to set up JAAS configuration and to create tables with some users in a database. In this case I will create two tables, one with the roles (MyAdmin role) and one with the users and passwords.
Doing that is simple.

```
CREATE TABLE Users(username VARCHAR(64) PRIMARY KEY, password VARCHAR(64))
CREATE TABLE UserRoles(username VARCHAR(64), userRoles VARCHAR(32))
INSERT INTO Users VALUES ('oracle9i','123')
INSERT INTO UserRoles VALUES ('oracle9i','MyAdmin')
```

After that we need to configure some files in JBoss. The first file is **login-config.xml**. This file can be found in **server/default/conf** directory and the fragment of code that need to be added is :

```
<application-policy name="mailclient">
<authentication>
```

```
<login-module code="org.jboss.security.auth.spi.DatabaseServerLoginModule" flag="required">
<module-option name="dsJndiName">java:/XAOracleDS</module-option>
<module-option name="principalsQuery">select password from Users where username=?</module-
option>
<module-option name="rolesQuery">select userRoles,'Roles' from UserRoles where
username=?</module-option>
</login-module>
</authentication>
</application-policy>
```

This fragment of code simply uses application policy called "mailclient" (name of this application policy) with ORACLE DataSource and the data from two tables (users and userroles).

Other files that we need belong to the application and go together in a package. They will be explained later. The server has to be restart.

Application structure

Application structure is shown in the **Figure 41**. There is nothing much to say about the structure since it follows already shows rules.

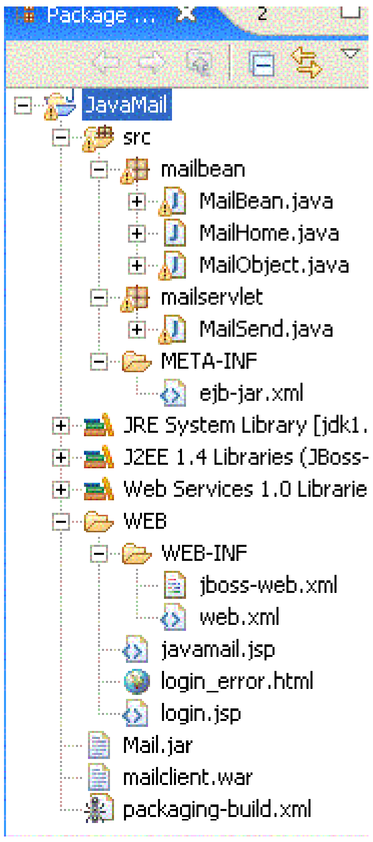


Figure 41: JavaMail structure

After running packaging and deployment two files will be created, **Mail.jar** which is an EJB that contains the code for sending email and **mailclient.war** which contains JSPs and Servlet.

The code

The code that sends email belongs to **javax.mail** library and is a part of JavaMail API. EJB method that sends email is "**public boolean send(String to,String from,String subject,String message)throws RemoteException**". It is in the **MailBean**.
JavaMail code looks like this :

```
            Session session = null;


            try {



               Message m = new MimeMessage(session);
               m.setFromnew InternetAddress(from));

               m.setRecipients(RecipientType.TO,InternetAddress.parse(to,
false));
               m.setSubject(subject);
               m.setSentDate(new Date());
               m.setContent(message,"text/plain");

               Transport.send(m);
            } catch (javax.mail.MessagingException e) {
               e.printStackTrace();
            }

            return true;
         }
```

Probably the most important code is in WEB-INF. There are two descriptors inside. The first belongs to JAAS policy and defines policy called "mailclient". It is called **jboss-web.xml.**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss-web PUBLIC "-//JBoss//DTD Web Application 2.3//EN"
"http://www.jboss.org/j2ee/dtd/jboss-web_3_2.dtd">

<jboss-web>
      <context-root>/mailclient</context-root>
```

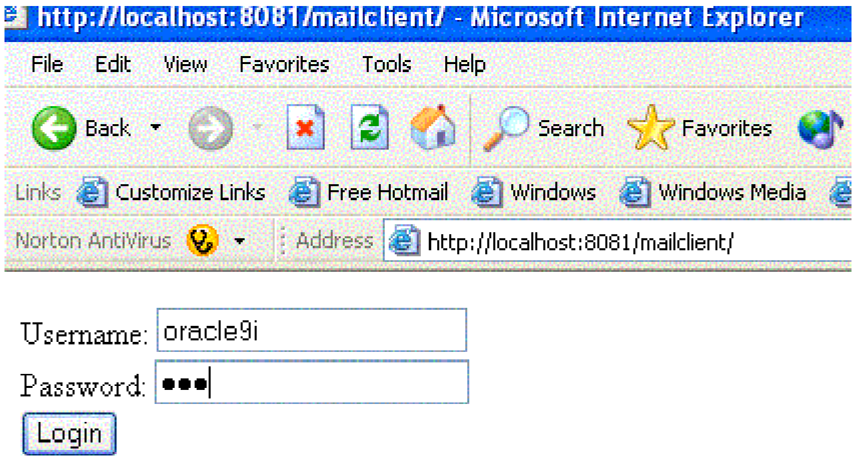
```
   <!-- Resource references -->
   <!-- Uncomment the security-domain to enable security. You will
      need to edit the htmladaptor login configuration to setup the
      login modules used to authentication users.
      -->
      <security-domain>java:/jaas/mailclient</security-domain>


</jboss-web>
```

The second descriptor is web.xml and it is used to describe servlet and JSP (optional). There are some new elements inside also:

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee" version="2.4"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <display-name>JBoss</display-name>


    <servlet>
        <display-name>MailSend</display-name>
        <servlet-name>MailSend</servlet-name>
        <servlet-class>mailservlet.MailSend</servlet-class>
    </servlet>

   <servlet>
     <display-name>login</display-name>
     <servlet-name>login</servlet-name>
     <jsp-file>/login.jsp</jsp-file>
   </servlet>

  <servlet>
    <display-name>javamail</display-name>
    <servlet-name>javamail</servlet-name>
    <jsp-file>/javamail.jsp</jsp-file>
  </servlet>



  <servlet-mapping>
     <servlet-name>MailSend</servlet-name>
     <url-pattern>/send</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>login</servlet-name>
    <url-pattern>/login.jsp</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>javamail</servlet-name>
    <url-pattern>/javamail.jsp</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
This fragment shows the first page that will be showed
    <welcome-file>/javamail.jsp</welcome-file>
  </welcome-file-list>

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>mailclient</web-resource-name>
      <description>An example security config that only allows users
with the role JBossAdmin to access the JavaMail web application
      </description>
      <url-pattern>/*</url-pattern> protect all resources
      <http-method>GET</http-method>
      <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
      <role-name>myAdmin</role-name>
    </auth-constraint>
  </security-constraint>


  <login-config> Login page is showed first and if authentication fails
login_error
    <auth-method>FORM</auth-method>
            <form-login-config>
                  <form-login-page>/login.jsp</form-login-page>
                  <form-error-page>/login_error.html</form-error-page>
          </form-login-config>

  </login-config>

  <security-role>
    <role-name>myAdmin</role-name>
  </security-role>

 </web-app>
```

Running the application

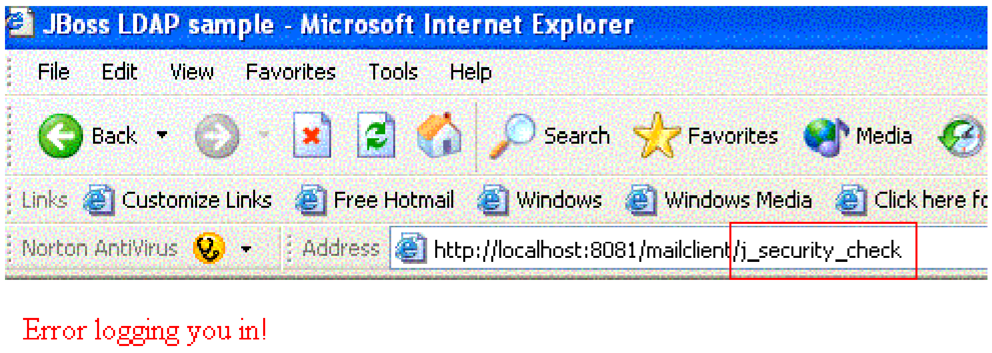
After going to **localhost:8081/mailclient** we can run the application. The next few snapshots show all-important moments:


http://localhost:8081/mailclient/ - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Links Customize Links Free Hotmail Windows Windows Media
Norton AntiVirus Address http://localhost:8081/mailclient/
Login

Username: oracle9i

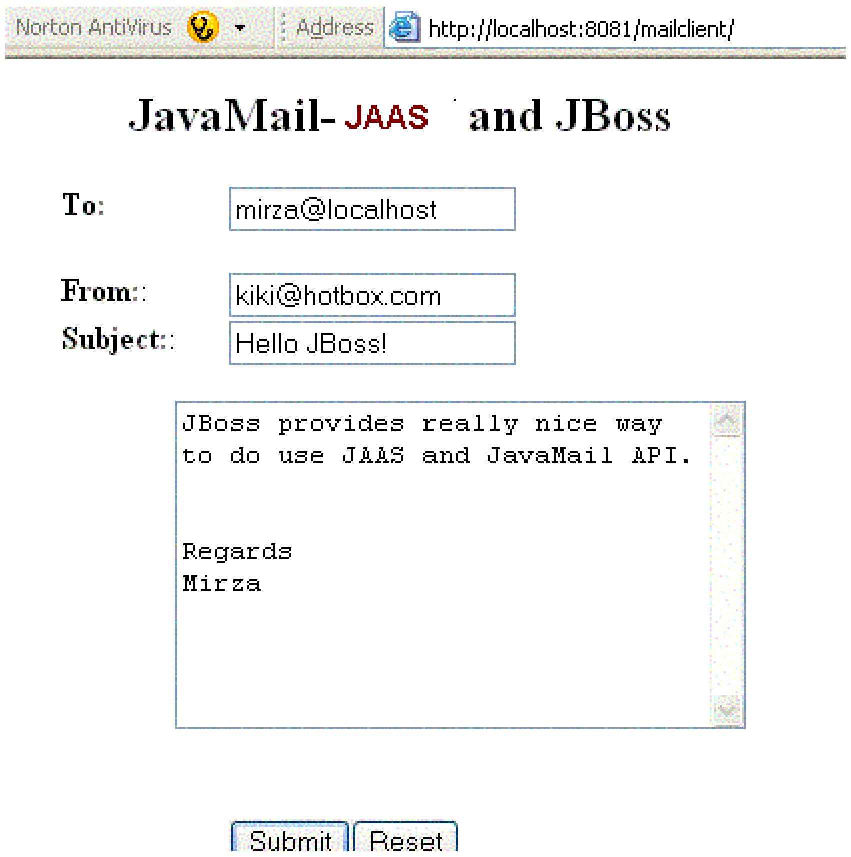
Password: ●●●

Figure 43: Log in the web


JBoss LDAP sample - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Customize Links Free Hotmail Windows Windows Media Click here fo
Norton AntiVirus Address http://localhost:8081/mailclient/j_security_check

Error logging you in!

Figure 44: Wrong password



http://localhost:8081/mailclient/

JavaMail- JAAS and JBoss

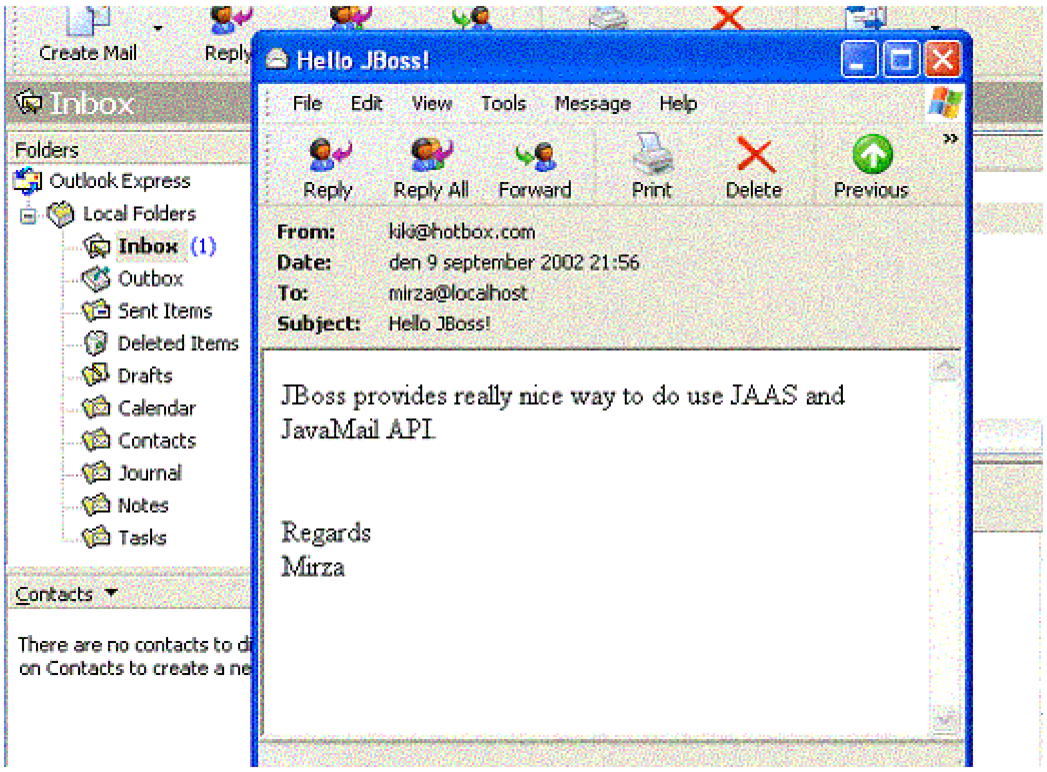
To: mirza@localhost

From:: kiki@hotbox.com

Subject:: Hello JBoss!

```
JBoss provides really nice way
to do use JAAS and JavaMail API.


Regards
Mirza
```

Figure 44: The login succeeded



There are no contacts to di
on Contacts to create a ne

From: kiki@hotbox.com
Date: den 9 september 2002 21:56
To: mirza@localhost
Subject: Hello JBoss!

JBoss provides really nice way to do use JAAS and JavaMail API

Regards
Mirza

Figure 45: Bla Bla Bla

PART 3

Building with Ant

Apart from using Eclipse for packaging and deployment, Ant program can also be used separately. That basically means: downloading Ant from the www.apache.org ,installing it , creating "build-file" and at the end simply running it. Probably the most difficult part of this is creating "build-file".

Build

Build file is simply and **xml** file that contains set of paths to the compiler and server, set of properties (folders that contains our files) and set of targets (description of various actions). It can be very difficult to create a good "build-file" for packaging and deployment of an application if the application is large. My example shows a build-file from the previous project "Student". That build-file creates 3 extra files: MyBean.jar (contains EJBs and descriptors), web-client.war (contains Servlets) and enterprise.ear (contains both of them).

The most important parts of the build-file that I call **student-build** are:

1. Describes paths and properties. This part contains path for **jboss-build.properties** file which simply describes all important paths of the server JBoss. I also define two properties for this file: src (source code), build 8compiled code).

```
<project name="student" default="all" basedir=".">
   <property file="../jboss-build.properties"/>


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



   <!-- The classpath for running the client -->

   <path id="client.classpath">
      <pathelement location="${build.dir}"/>
      <fileset dir="${jboss.home}/client">
```

```
            <include name="**/*.jar"/>
        </fileset>
    </path>


   <!-- The build classpath -->
    <path id="build.classpath">
        <path refid="client.classpath"/>
            <fileset dir="${jboss.server}/lib/">
            <include name="javax.servlet*.jar"/>
        </fileset>
        <!-- for java2wsdl -->
            <fileset dir="${jboss.server}/deploy/jboss-ws4ee.sar/">
            <include name="*.jar"/>
        </fileset>
    </path>
```

2. Set of targets for creation of the most basic packages : MyBean.jar, web-client.war and enterprise.ear. Each of them contains its descriptors.

```
    <!--
===================================================================
==== -->
    <!-- Initialises the build.                              -->
    <!--
===================================================================
==== -->
    <target name="prepare">
        <mkdir dir="${build.dir}"/>
    </target>

    <!--
===================================================================
= -->
    <!-- Compiles the source code                                -->
    <!--
===================================================================
= -->
    <target name="compile" depends="prepare">
        <javac destdir="${build.dir}" classpathref="build.classpath"
               debug="on">
            <src path="${src.dir}"/>
        </javac>
    </target>
```

```
   <target name="MyBean" depends="compile">
      <mkdir dir="jar" />
      <delete file="jar/MyBean.jar"/>

      <jar jarfile="jar/MyBean.jar">
         <metainf dir="src/meta-beans" includes="**/*.xml" />

         <fileset dir="${build.dir}">
            <include name="statefulEJBA/**"/>
            <include name="statefulEJBB/**" />
            <include name="studentEJBFactory/**" />
            <include name="statelessEJBPattern/**" />
            <include name="student/**" />
         </fileset>
      </jar>
   </target>

  <target name="servlet" depends="compile">
      <mkdir dir="jar" />
      <delete file="jar/web-client.war"/>

  <war warfile="jar/web-client.war" webxml="WEB-INF/web.xml">

      <fileset dir="${build.dir}">
        <include name="studentclient/*.class"/>
      </fileset>


   </war>
  </target>





 <!-- Creates an ear file containing the ejb jars and the web client war. -->
   <target name="enterprise">
      <delete file="jar/StudentEnterprise.ear"/>
      <ear destfile="jar/StudentEnterprise.ear" appxml="src/WEB-
INF/application.xml">
         <fileset dir="jar" includes="*.jar,*.war"/>

      </ear>
   </target>

   <!-- Deploys the EAR file by copying it to the JBoss deploy directory.  -->
   <target name="deploy" depends="enterprise">
      <copy file="jar/StudentEnterprise.ear" todir="${jboss.server}/deploy"/>
   </target>


 <!-- Deletes build directory -->
  <target name="clean">
     <delete dir="${build.dir}" />
     <mkdir dir="${build.dir}"  />


   </target>


</project>
```

Running the **student-build.xml** file is easy. Let say to package EJBs we should run "**ant student-bean.xml MyBean". MyBean** is a target in this case. To package servlets we should run target called "servlet", "**ant student-bean.xml servelt",** and to package those two into an enterprise package we should run "**ant student-bean.xml enterprise".** Deployment should use target **deploy.**

In order to package all components into single package (*.ear) we must create **application.xml** file. This file has been discussed before and in this example it looks something like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <application xmlns="http://java.sun.com/xml/ns/j2ee" version="1.4"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com /xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/application_1_4.xsd">
  <display-name>JBossDukesBank</display-name>
  <description>Application description</description>
- <module>
  <ejb>MyBean.jar</ejb>
  </module>
- <web>
  <web-uri>web-client.war</web-uri>
  <context-root>web-client</context-root>
  </web>
  </module>
- <security-role>

  </application>
```

After running those three targets, three packages should be created.

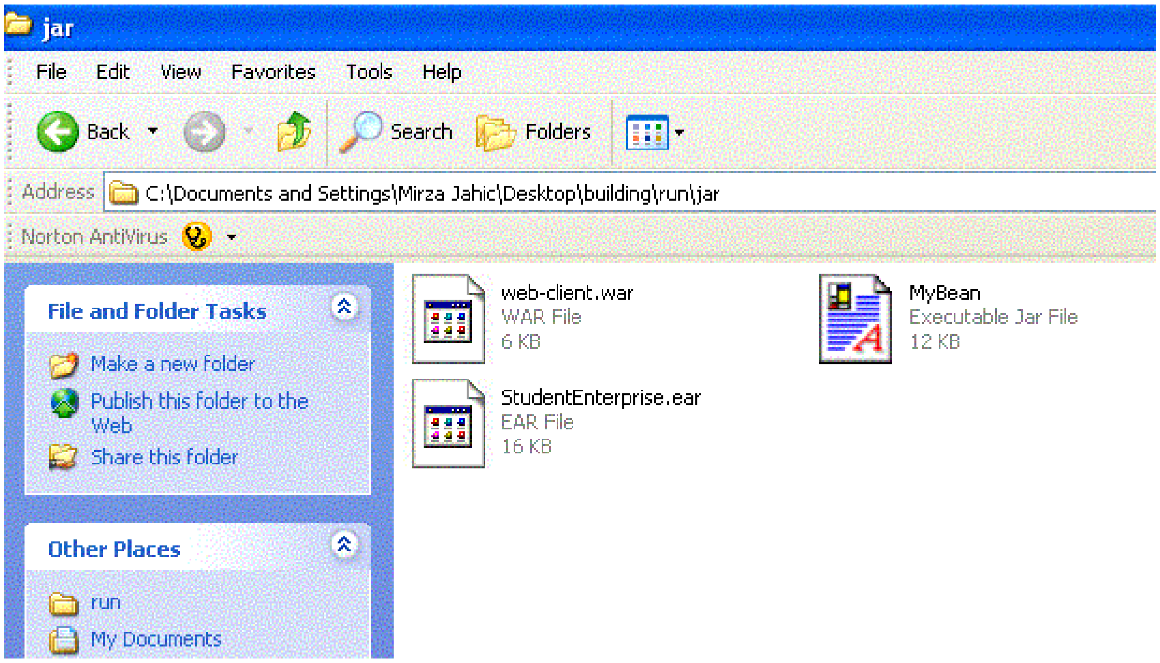


Figure 46: Packaging with Ant

The file that describes properties of the JBoss is called **jboss-build.xml**. This file looks like this:

```
# Set the path to the JBoss directory containing the JBoss application server
# (This is the one containing directories like "bin", "client" etc.)

jboss.home=C:/JBoss
jboss.server=${jboss.home}/server/default
jboss.deploy.dir=${jboss.server}/deploy
```

Deployment with Ant can be necessary in situation where the team of developers is large and the number of files too. In that case one developer can package and deploy everything from one place.

XDoclets

Creating the J2EE components in a conventional way can be tedious task. Sometimes applications are huge and generating descriptors can be error prone. In order to reduce errors and to make development faster and cheaper XDoclets are used. The are simply meta-data that can be used for generating home and remote interface of EJB and all descriptors that we might need. Eclipse makes generation of XDoclets very easy and the developer simply creates only a Bean with its business methods. After that business method is described by XDoclet and based on that all interfaces and descriptors for the given EJB are generated at once. Running XDoclet is similar to running packaging from the property menu.

Some examples of XDoclets :

```
*
* @ejb.bean name = "Fibo"
* display-name = "Fibo EJB"
* description = "EJB that computes Fibonacci suite"
* view-type = "remote"
* jndi-name = "ejb/tutorial/Fibo"
*/
public class FiboBean implements SessionBean {


/**
* @param number
* @return
*
* @ejb.interface-method view-type = "remote"
*/
public double[] compute(int number) {
```

Conclusion

JBoss is far from simple application server to use. It is probably one of the most difficult application servers but also one of the best and the most popular. In order to reduce the time of working with JBoss and to make it easier Eclipse and JBoss-IDE are used. They help in developing applications faster and using JBoss easier. Ant is also very important tool that comes together with Eclipse but can also be used separately.

References:

- **"Start guide" www.Jboss.com**
- **JBoss-IDE by Hans Dockter and Laurent Etiemble**
- **www.eclipse.org**
- **www.Apache.org/Ant**