

The Following information and code is compiled by Mr. Faisal Hassan (Lecturer in CCIT) for Advance Java Technologies.

## Container Managed Persistence (CMP) EJB

This is an example of a JSP-page calling an Entity Bean that uses container managed persistence (CMP). The result looks like this:

If you have read my former mini example about the time bean [☞](#), then you already know that can't spell. I hope you can understand it anyway.

Some of the things in this example have already been explained in the time bean example, so in this example I will not discuss those things again.

In this example, I have not used any of the things making EJB development easier, like Eclipse, Ant or Xdoclet. I have tried to strip of everything just to make this example as easy as possible to understand. This is an example using geronimo 1.1, java 1.4 and EJB 2.1 along with the in geronimo built in database called derby. In the end I'll show how to switch database to MySQL.

Entity beans seems to be on their way out. In the spec for EJB 3.0 they do not exist any more. In java EE (1.5) there is a new API called Java Persistence API. I guess that it is more like object relational mapping stuff such as Hibernate. But I have not looked in to it yet so I don't know.

Anyhow, I have a table in my derby database that I called phonebook. I started by using the DBManager in the geronimo console ( [http://localhost:8080/console/portal/internalDB/internalDB\\_DBManager](http://localhost:8080/console/portal/internalDB/internalDB_DBManager) [☞](#) ) And created a new database that I called PhonebookDB.

Then I created the table and put some values in the table using these SQL statements.

```
create table phonebook (name varchar(15) primary key, number varchar(15)) ;
insert into phonebook values('Mattias', '1234');
insert into phonebook values('Pamela', '5678');
```

This EJB application searches for numbers when the user enters names.

An entity java bean, as you know, represents a row in the database table. An entity java bean only have getter and setter methods for the fields in the table. It does not have properties (variables) and it does not have any other methods than the getter/setter and methods in the EntityBean interface (note that by putting it this way I did not have to write that entity beans can't have business methods as I refuse to refer to so called business methods with the word business methods. Let's not have advertising people controlling our way of talking and by that also our way of thinking, shall we. And also I must admit, that entity EJB can, but should not, have methods that I refuse to refer to as business methods). The entity bean is also always abstract.

This is my entity bean that I have put in a package named myphonebookpak:

```

package myphonebookpak;

import javax.ejb.*;


public abstract class MyPhonebookBean implements EntityBean {

    public void ejbLoad() {}
    public void ejbStore() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void setEntityContext(EntityContext ctx) {}
    public void unsetEntityContext() {}
    public void ejbRemove() throws RemoveException {}
    public void ejbPostCreate(String name, String number) throws javax.ejb.CreateException {}
    public String ejbCreate(String name, String number) throws javax.ejb.CreateException {
        setName(name);
        setNumber(number);
        return null; // ejbCreate must return null according to ejb spec. Why? Thats what asked to...
    }

    public abstract String getName() ;
    public abstract void setName(String name) ;
    public abstract String getNumber() ;
    public abstract void setNumber(String number) ;

}

```

The fist methods are part of the EntityBean interface. They are never used for any good as far as I know and are only there to mess up your code. ejbCreate is a constructor, and when you craete a bean it is stored in the database. There are many pages out there that explains the difference between ejbCreate and ejbPostCreate. This text below is cited from <http://www.jguru.com/faq/view.jsp?EID=1036904> 

-----<@-----  
 ejbCreate() is called before the state of the bean is written to the persistence storage (database). After this method is completed, a new record (based on the persistence fields) is created and written. If the Entity EJB is BMP, then this method must contain the code for writing the new record to the persistence storage. If you are developing an EJB following 2.0 specs, you can have overloading methods in the form of ejbCreateXXX(). This will improve the development so you can have different behaviour for creating a bean, if the parameters differs. The only requirement is that for each ejbCreateXXX() you need to have corrsponding createXXX() methods in the home or local interface.

ejbPostCreate() is called after the bean has been written to the database and the bean data has been assigned to an EJB object, so when the bean is available. In an CMP Entity EJB, this method is normally used to manage the beans' container-managed relationship fields.

----->@-----  
 Note the talk about ejbCreateXXX if you don't have all the parameters needed for inserating a new row with all fileds in a table, if I understand it right.

Entity beans have home and remote interfaces. In this example I only call the bean from a JSP-page in the same application server so I use the localhome and local interfaces instead that don't make use of the network.

This is the code for the local home interface called MyPhonebookLocalHome :

```
package myphonebookpak;
import javax.ejb.*;
public interface MyPhonebookLocalHome extends EJBLocalHome {
    public MyPhonebookLocal create() throws javax.ejb.CreateException;
    public MyPhonebookLocal findByPrimaryKey(String pk) throws FinderException;
}
```

The findByPrimaryKey is like a SQL select statement (Select \* from phonebook where name = ?).

This is the local interface called MyPhonebookLocal:

```
package myphonebookpak;
import javax.ejb.*;
public interface MyPhonebookLocal extends EJBLocalObject
{
    public String getName();
    public void setName(String name);
    public String getNumber();
    public void setNumber(String number);
}
```

The set-methods are like update statements and the get-methods are like rs.getString("name") if you know what I mean.

Now, lets have a look at the deployment descriptors. You need 2 of them, ejb-jar.xml and openejb-jar.xml. This is what they look like:

ejb-jar.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee"
    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/ejb-jar_2_1.xsd"
    version="2.1">
    <enterprise-beans>
        <entity>
            <description>An EJB named MyPhonebookBean</description>
            <ejb-name>MyPhonebookBean</ejb-name>
            <local-home>myphonebookpak.MyPhonebookLocalHome</local-home>
            <local>myphonebookpak.MyPhonebookLocal</local>
            <ejb-class>myphonebookpak.MyPhonebookBean</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>
            <cmp-field>
```

```

    <description>A persons name</description>
    <field-name>name</field-name>
  </cmp-field>
  <cmp-field>
    <description>A persons phone number</description>
    <field-name>number</field-name>
  </cmp-field>
  <primkey-field>name</primkey-field>
</entity>
</enterprise-beans>
</ejb-jar>

```

You recognize the tags `ejb-name`, `local-home` etc from the time bean example. But then we have some new tags, they are stright forward describing the the non existing properties (variables) that we have getter/setter mtehdos for. Also the primary key is pointed out. Here comes the `openejb-jar.xml` file:

```

<?xml version="1.0" encoding="UTF-8"?>
<openejb-jar xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1"
  xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.1"
  xmlns:pkgen="http://www.openejb.org/xml/ns/pkggen-2.0"
  xmlns:sec="http://geronimo.apache.org/xml/ns/security-1.1"
  xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.1">
  <sys:environment>
    <sys:moduleId>
      <sys:groupId>default</sys:groupId>
      <sys:artifactId>MyPhonebookBean_openejb_artifactId</sys:artifactId>
      <sys:version>1.0</sys:version>
      <sys:type>car</sys:type>
    </sys:moduleId>
  </sys:environment>

    <cmp-connection-factory>
      <resource-link>PhonebookPool</resource-link>
    </cmp-connection-factory>

  <enterprise-beans>
    <entity>
      <ejb-name>MyPhonebookBean</ejb-name>
      <local-jndi-name>MyPhonebookBean</local-jndi-name>
      <table-name>phonebook</table-name>
      <cmp-field-mapping>
        <cmp-field-name>name</cmp-field-name>
        <table-column>NAME</table-column>
      </cmp-field-mapping>
      <cmp-field-mapping>
        <cmp-field-name>number</cmp-field-name>
        <table-column>NUMBER</table-column>
      </cmp-field-mapping>
    </entity>
  </enterprise-beans>
</openejb-jar>

```

Note that this does not have `ejb-ref`, `ref-name` or `ejb-link` tags. Instead there are the tags `ejb-name` and `local-jndi-name`. Then there is mappings between the bean and the underlying table in the database.

That's it. Now over to the JSP page:

```
{
<%@ page contentType="text/html" import="myphonebookpak.*, javax.naming.* " %>

<%
    String searchName = "";
    if (request.getParameter("searchname") != null) {
        searchName=request.getParameter("searchname");
    }
%>

<html><head><title>Phonebook</title></head><body>
<form action="index.jsp">
<b>Search number</b><br>
Enter name: <input type="text" name="searchname" value="<%=searchName%>">
<input type="submit" value="Search">
(Test with <a href="index.jsp?searchname=Mattias">Mattias</a>)
</form>
<%
    if (! searchName.equals("")) {
        String number="";
        try {
            Context context = new InitialContext();
            MyPhonebookLocalHome myPhonebookHomeLocal =
(MyPhonebookLocalHome)context.lookup("java:comp/env/ejb/MyPhonebookBean");
            MyPhonebookLocal myPhonebookLocal =
myPhonebookHomeLocal.findByPrimaryKey(searchName);
            number = myPhonebookLocal.getNumber();
        }
        catch (Exception e) {
            number=e.toString();
        }
        out.println("This is the number returned from the EJB when seachring for '"+
searchName+"' : " + number);
    }
%>
</body></html>
```

Just to make this JSP page interactive so you can search for different names I have made it just a little bit more complicated, but this way is more fun. The important part is:

```
{
    MyPhonebookLocalHome myPhonebookHomeLocal =
(MyPhonebookLocalHome)context.lookup("java:comp/env/ejb/MyPhonebookBean");
    MyPhonebookLocal myPhonebookLocal = myPhonebookHomeLocal.findByPrimaryKey(searchName);
    number = myPhonebookLocal.getNumber();
}
```

First there is the lookup MyPhonebookBean in "java:comp/env/ejb/MyPhonebookBean" that corresponds to the local-jndi-name in the openejb-jar.xml file. findByPrimaryKey is like a select statement and myPhonebookLocal.getNumber() simply gets the number from the EJB that we just found.

Deployment descriptors:

geronimo-web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1"
  xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.1"
  xmlns:sec="http://geronimo.apache.org/xml/ns/security-1.1"
  xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.1">
  <sys:environment>
    <sys:moduleId>
      <sys:groupId>default</sys:groupId>
      <sys:artifactId>MyPhonebookWeb</sys:artifactId>
      <sys:version>1.0</sys:version>
      <sys:type>car</sys:type>
    </sys:moduleId>
  </sys:environment>
  <context-root>/myphonebook</context-root>
</web-app>
```

Only important here (as far as I understand) is the context-root that becomes the URI.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  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>
    MyPhonebookWeb</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <ejb-local-ref>
    <ejb-ref-name>ejb/MyPhonebookBean</ejb-ref-name>
    <ejb-ref-type>Entity</ejb-ref-type>
    <local-home>myphonebookpak.MyPhonebookLocalHome</local-home>
    <local>myphonebookpak.MyPhonebookLocal</local>
  </ejb-local-ref>
</web-app>
```

This is similar to what we saw in the former time bean example.

Finally there is the deployment descriptors for the EAR-file, that in this case also includes a database pool xml-file. The best thing to do is to start with the database pool. In Geronimo console there a link in the left navigation called "Database Pools". From there, click the link "Using the Geronimo database pool wizard". You will end up here:

http://localhost:8080/console/portal/services/services\_jdbc/  
\_rp\_services\_jdbc\_row1\_col1\_p1\_adapterDisplayName/  
1\_TranQL0x8Generic0x8JDBC0x8Resource0x8Adapter/\_rp\_services\_jdbc\_row1\_col1\_p1\_mode  
/1\_rdbms/  
\_pm\_services\_jdbc\_row1\_col1\_p1/view/\_st\_services\_jdbc\_row1\_col1\_p1/normal/\_pid/  
services\_jdbc\_row1\_col1\_p1/\_md\_services\_jdbc\_row1\_col1\_p1/view

Follow the wizard using the derby database click on the test button to see that it works, and then you click the button "Skip test and show plan". Then geronimo, this is very handy, show a complete database pool xml file, and also instructions of how to use it under the section "Add to EAR"!

This is what my pool file derby\_PhonebookPool.xml looks like. Note that I in the bottom of the file added a comment with the text from the wizard telling me how to use the pool file.

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.1">
  <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
    <dep:moduleId>
      <dep:groupId>console.dbpool</dep:groupId>
      <dep:artifactId>PhonebookPool</dep:artifactId>
      <dep:version>1.0</dep:version>
      <dep:type>rar</dep:type>
    </dep:moduleId>
    <dep:dependencies>
      <dep:dependency>
        <dep:groupId>org.apache.derby</dep:groupId>
        <dep:artifactId>derby</dep:artifactId>
        <dep:version>10.1.1.0</dep:version>
        <dep:type>jar</dep:type>
      </dep:dependency>
    </dep:dependencies>
  </dep:environment>
  <resourceadapter>
    <outbound-resourceadapter>
      <connection-definition>
        <connectionfactory-interface>javax.sql.DataSource</connectionfactory-interface>
        <connectiondefinition-instance>
          <name>PhonebookPool</name>

          <config-property-setting name="Driver">org.apache.derby.jdbc.EmbeddedDriver</config-property-
setting>
          <config-property-setting name="UserName">app</config-property-setting>
          <config-property-setting name="ConnectionURL">jdbc:derby:PhonebookDB</config-property-setting>
        </connectiondefinition-instance>
      </outbound-resourceadapter>
    </resourceadapter>
  </connector>
```

```

    </connection-definition>
  </outbound-resourceadapter>
</resourceadapter>
</connector>

```

```
<!--
```

Here is a comment with the text from the wizard explaining how to use this pool file

Add to EAR: Instead of deploying as a top-level database pool, you can deploy this pool as part of an EAR.

To add a database pool to an EAR using this plan:

Copy and paste the plan to a file

Save the plan file to the top level of your EAR

Copy the RAR file from GERONIMO\_HOME\repository\tranql\tranql-connector\1.2\tranql-connector-1.2.rar to the top level of your EAR

Create a META-INF/geronimo-application.xml file in your EAR that has a module entry like this

(substituting the correct RAR file name and plan file name):

```

<application
  xmlns="http://geronimo.apache.org/xml/ns/j2ee/application-1.0"
  configId="MyApplication">
  <module>
    <connector>rar-file-name.rar</connector>
    <alt-dd>plan-file-name.xml</alt-dd>
  </module>
</application>
-->

```

It seems as "app" is a good user name to use with the build in derby database. No password.

But this can perhaps be configured somewhere. Following the instructions from the wizard I made a copy of tranql-connector-1.2.rar and saved it in the applications home directory (see file layout later on) along with the pool file.

There is two more deployment descriptors left. The application.xml and the geronimo-application.xml

application.xml:

```

<?xml version="1.0" encoding="UTF-8"?>
<application id="Application_ID" version="1.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  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>MyphonebookEJBApp</display-name>
  <module>
    <ejb>myphonebook-ejb.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>myphonebook-web.war</web-uri>
      <context-root>/myphonebook</context-root>
    </web>
  </module>
  <module>
    <connector>tranql-connector-1.2.rar</connector>
  </module>
</application>

```

There are three modules. The ejb and web modules we recognize from the time bean example, but the connector part is new. This simply states that we have a connector. I am not sure exactly what tranql-connector really is, but I handles the contact with the database in some way that we perhaps do not need to understand.

geronimo-application.xml As you can see here we also have a module conector tag. Here the pool file is referenced. Not that in derby\_PhonebookPool.xml there is a name-tag called PhonebookPool that is referenced in the resource-link in openejb-jar.xml file. This is how the applications gets to know about the database.

This is the file layout (If you want to do this kind of layouts and are on a windows system like me, try the DOS command tree /F):

```
App-home
^|
^| derby_PhonebookPool.xml
^| tranql-connector-1.2.rar
^|
+---ejb
^| +---META-INF
^| ^|   ejb-jar.xml
^| ^|   openejb-jar.xml
^| ^|
^| +---myphonebookpak
^|     MyPhonebookBean.java
^|     MyPhonebookLocal.java
^|     MyPhonebookLocalHome.java
^|
+---META-INF
^|   application.xml
^|   geronimo-application.xml
^|
+---web
^|   index.jsp
^|
+---WEB-INF
^|   geronimo-web.xml
^|   web.xml
^|
+---classes
^|   +---myphonebookpak
^|         MyPhonebookLocal.class
^|         MyPhonebookLocalHome.class
}
```

In order to compile this application we start with the EJB. Then, when it is compiled, we copy MyPhonebookLocal.class and MyPhonebookLocalHome.class to the web\WEB-INF\classes\myphonebookpak, they are needed there or else the JSP-page will not work. When compilations is done, it is time to pack the application, that is done in three steps, first pack the EJB, then the WAR (that is the JSP-page) and finally pack it all together in an EAR-file. This is the steps for compiling and packing the application, assuming you are in the applications home directory (in my case under c:\java\j2ee\phone so I start with cd c:\java\j2ee\phone). I

have put this commands in a bat file (sh file for linux users) so I don't have to do it by hand evry time. But really, one should use Ant for this kind of work instead.

```
set CLPATH=C:\geronimo\geronimo-1.1\repository\org\apache\geronimo\specs\geronimo-  
ejb_2.1_spec\1.0.1\geronimo-ejb_2.1_spec-1.0.1.jar  
cd ejb  
javac -classpath %CLPATH% myphonebookpak\*.java  
copy myphonebookpak\MyphonebookLocal.class ..\web\WEB-INF\classes\myphonebookpak\  
copy myphonebookpak\MyphonebookLocalHome.class ..\web\WEB-INF\classes\myphonebookpak\  
jar -cf ../myphonebook-ejb.jar myphonebookpak META-INF  
cd ..\web  
jar -cf ../myphonebook-web.war index.jsp WEB-INF  
cd ..  
jar -cf myphonebook-ear.ear myphonebook-ejb.jar myphonebook-web.war derby_PhonebookPool.xml tranql-  
connector-1.2.rar META-INF
```

This is very similar to the time bean example, but here we also add the two files derby\_PhonebookPool.xml and tranql-connector-1.2.rar to the ear-file.

To deploy this I use (but you can of course also use the geronimo console) the deploy tool in the geronimo servers bin directory. Like this on my windows system where I have geonimo installed under c:\geronimo\geronimo-1.1 :

```
c:\geronimo\geronimo-1.1\bin\deploy.bat --user system --password manager deploy  
myphonebook-ear.ear
```

Now you can look at the application at: <http://localhost:8080/myphonebook> 🌐

If you change things, compile and pack evrything again and change the above deployment command to redeploy myphonebook-ear.ear in the end of the line.

All soure code can be downloaded in the zip-file: [myphonebook.zip](#) 📄

## Using MySQL as a database instead of derby

I have a MySQL 5.0.24-community-nt database installed on my computer. Let's see what we need to do to get the phnoebook example work with MySQL instead!

First set up the database! Logg in as root and create a new database called PhonebookDB (same as for derby earlier in this example). Then create the table and put som values. Be sure to put in a new person in this table so we later on can prove to ourselves that we are really using the new mysql database instead of the old derby. This is my commands:

```
c:\>mysql -u root -p  
Enter password: *****  
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 1 to server version: 5.0.24-community-nt  
  
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.  
  
mysql> create database PhonebookDB;  
mysql> use PhonebookDB;  
mysql> create table phonebook (name varchar(15) primary key, number varchar(15)) ;
```

```

mysql> insert into phonebook values('Mattias', '1234');
mysql> insert into phonebook values('Pamela', '5678');
mysql> insert into phonebook values('Cindy', '906090');
mysql> GRANT ALL PRIVILEGES ON PhonebookDB.* TO phone@localhost IDENTIFIED BY 'book' WITH GRANT
OPTION;
mysql> FLUSH PRIVILEGES;

```

Now, let's use the pool wizard again as described earlier, but this time, make a mysql pool by selecting "Database Type": MySQL. I can't remember if the JDBC driver comes default or if you would have to download it by simply clicking the button "Download driver". Fill in "phone" as user name and "book" as password.

You should end up with a database pool file like this. I save it with the name mysql\_PhonebookPool.xml in the same directory as the former derby\_PhonebookPool.xml :

```

<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.1">
  <dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
    <dep:moduleId>
      <dep:groupId>console.dbpool</dep:groupId>
      <dep:artifactId>PhonebookPool</dep:artifactId>
      <dep:version>1.0</dep:version>
      <dep:type>rar</dep:type>
    </dep:moduleId>
    <dep:dependencies>
      <dep:dependency>
        <dep:groupId>mysql</dep:groupId>
        <dep:artifactId>mysql-connector-java</dep:artifactId>
        <dep:version>3.1.12</dep:version>
        <dep:type>jar</dep:type>
      </dep:dependency>
    </dep:dependencies>
  </dep:environment>
  <resourceadapter>
    <outbound-resourceadapter>
      <connection-definition>
        <connectionfactory-interface>javax.sql.DataSource</connectionfactory-interface>
        <connectiondefinition-instance>
          <name>PhonebookPool</name>
          <config-property-setting name="Password">book</config-property-setting>
          <config-property-setting name="Driver">com.mysql.jdbc.Driver</config-property-setting>
          <config-property-setting name="UserName">phone</config-property-setting>
          <config-property-setting
name="ConnectionURL">jdbc:mysql://localhost:3306/PhonebookDB</config-property-setting>
          <connectionmanager>
            <local-transaction/>
            <single-pool>
              <max-size>10</max-size>
              <min-size>0</min-size>
              <match-one/>
            </single-pool>
          </connectionmanager>
        </connectiondefinition-instance>
      </connection-definition>
    </outbound-resourceadapter>
  </resourceadapter>
</connector>

```

```
        </connectionmanager>
    </connectiondefinition-instance>
</connection-definition>
</outbound-resourceadapter>
</resourceadapter>
</connector>
```

All we have to do now is to change the pool file reference in the geronimo-application.xml file so that the module part looks like this instead.

```
<module>
  <connector>tranql-connector-1.2.rar</connector>
  <alt-dd>mysql_PhonebookPool.xml</alt-dd>
</module>
```

Now repack the ear file like this with the derby pool file switched for the corresponding mysql:  
jar -cf myphonebook-ear.ear myphonebook-ejb.jar myphonebook-web.war  
mysql\_PhonebookPool.xml tranql-connector-1.2.rar META-INF

Redeploy the application by the command :

```
c:\geronimo\geronimo-1.1\bin\deploy.bat --user system --password manager redeploy  
myphonebook-ear.ear
```

Now we should be able to find Cindy's phone number to prove that we are really using the mysql database. This picture shows this