

Instruction Sheet

Feb 10, 2005

Creating and using an Entity Bean in JRun4.

These instructions will guide you through the process of creating and using an Entity Bean on the Macromedia JRun4 Application Server.

1. Make sure you have Oracle accounts setup before you try going through this sheet.
2. Start your admin server and then open JMC.

Create a new 'Server' in JRun4 (optional)

When you create a new server, JRun creates a default web application and a blank EJB directory under that server.

1. Click on the "Create New Server" link on the top of the JMC window
2. Enter 'j2eeSamples' as the server name.
3. Create the server
4. Select two port numbers out of the range allocated to you and enter it in the JNDI Provider URL (localhost:99999) and Web Server Port Number (99999). You can leave the Web Connector Port Number as it is.
5. Click on "Update Port Numbers" and then click on "Finish"

Setup the directory structure and some naming tasks

We are going to make the following components:

- demo-ear: An enterprise application
 - entity-ejb: An entity bean inside demo-ear
 - entity-war: A web application (JSP) that connects to the entity bean
1. Under the \$JRUN/servers/j2eeSamples (*server-root* for short) directory, rename default-ear to demo-ear.
 2. Next, make changes to *server-root/demo-ear/META-INF/application.xml* file:
 - a. Modify the display name of the application, from "default-ear" to say, "Demo: J2EE Components". This is what you'll see in JMC.
 - b. Modify the web-uri from 'default-war' to say, 'entity-war'. That will be the actual name of our JSP web application. We will be changing the directory name as in step 3 below.
 - c. Modify the name of the ejb from 'default-ejb' to 'entity-ejb'. That will be the actual name of our entity bean. We will be changing the directory name as in step 3 below.
 - d. Provide a context root, say /entity. This is part of the URL after the server address and port number that is required to access the web application (entity-war).
 3. Under the directory *server-root/demo-ear* rename the directory default-war to entity-war and default-ejb to entity-ejb.

Create a table in Oracle

This table will not be used to store data by the JRun container. Currently, JRun does not support making changes in existing tables. The JRun container will create its own table corresponding to the entity bean created in your Oracle database account. We create this

table here so that you can use the fields in the JRun Deployment Wizard to create the bean itself.

1. Create a table named Diary with the following schema in Oracle:

ID	INT
NAME	VARCHAR(30)
PHONE	VARCHAR(20)

Setup datasource in JRun through JMC

1. Start the j2eeSamples server through JMC
2. Once it starts up, go into the 'Resources' section of the j2eeSamples server.
3. Under JDBC Data Sources, select the Database Driver as Oracle. Provide a Data Source Name, say 'MyOracleDB', and click on Add. Using this data source, you will be able to connect to your Oracle account.
4. Set Oracle SID to 'cseadb'
5. Set Server IP/Hostname to 'oraserve.cse.buffalo.edu'. This is the address of the machine on the CSE network on which the Oracle database resides.
6. The Server Port will be set to '1521'. You must leave it as it is.
7. Fill up the User Name, Password and Verify Password fields according to your Oracle username and password.
8. Then click on Submit. If all goes well, you should see MyOracleDB in the list of data sources.
9. You can click on the green check mark button to verify the connection with the database. You should see a message 'Connected to MyOracleDB successfully'.

Creating the Entity Bean Classes JRun Enterprise Deployment Wizard

1. Open the JRun Deployment Wizard (jrunwizard).
2. Click on the "New Project" button. In the JRun wizard, each project would ideally be particular to a single component that you create in your application. This identification though, has no effect on your actual components or application at all.
3. We name the project after the name of our ejb component. Call it entity-ejb. Sticklers may precede the name of the component with the server name.
4. Enter the source directory as *server-root/demo-ear/entity-ejb*. This is the path where the .java source files that pertain to the EJB are placed.
5. Enter the classes directory the same as above. This is the path where the .class compiled bytecode files are placed. Since this wizard also has the capability to use javac (the Java compiler) and compile the generated source files, this classes directory is prompted from the user.
6. Press Ok.
7. On the main jrunwizard screen, click on "Add New Bean Descriptor".
8. In the popup window, select Entity Bean - Container-Managed (CMP 2.x).
9. Press Ok.
10. Enter Package Name as say, 'demo'. Try to understand clearly how packages work in Java. Since we have a package called 'demo' for this bean, under the *server-root/demo-ear/entity-ejb* directory, there will now be a directory called demo, inside which the source files that are generated will be present. Also, for your application to work, you must have your class files also inside the demo directory.

So if you have to move your compiled class files somewhere for purposes that I will explain later, then you must copy the whole directory 'demo'. At all times, to use these files you must precede these classes by the package name (unless you 'import' these files in your program).

11. Specify EJB Name as say, 'Diary'.
12. Click on Bean Objects.
13. Check "Is Local" and uncheck "Is Remote". Here you can select whether the bean should be remote or local. In most cases, entity beans almost always need to be local. Listed here are the name of the interfaces and the name of the class that the wizard will create for you. You should maintain the naming convention.
14. Under the Bean-Specific section, select the Primary Key Class as java.lang.Integer. Note that this is not the primitive int, but the wrapper class Integer. This is the data type of the primary key.
15. Set the Abstract Schema Name to 'diary_abstract_schema'. This is the name of the table that JRun will create in your Oracle account. It will not touch the records in your original table 'Diary'. You will not be able to touch the records in 'diary_abstract_schema' through Oracle's sqlplus.
16. Set the Data Source JNDI Name to 'MyOracleDB'
17. Leave the other details under this section as is.
18. In the 'CMP' tab, check that the Data Source field is set to 'MyOracleDB'. Now select the Table from the list as 'DIARY'. The fields automatically get loaded in the grid below.
19. You will need to set the Field Type for the 'ID' field to 'Integer'. Check mark the 'In CMP' option for all the fields. (In CMP is to the left of the 'Param Type' column).
20. Check the 'PK' column for the field ID. Here you select that ID is the primary key column of this table. A method named findByPrimaryKey() will operate on this field. You can select here whether you want getter and setter methods for every field.
21. There is also a tab called CMP methods by which you can associate user-defined SQL queries to EJB methods. You can explore this option if you like.
22. Now click on the Save button.
23. And then click on 'Generate Source'. If all goes well, you should see a message saying source has been generated.
24. You can compile the source files through the wizard, but I prefer to compile them from the console.
25. Now you can close the wizard.

Compiling the source files

1. Make sure you have the following things setup properly:
 - a. You have the file \$JRUN/lib/jrun.jar in your CLASSPATH environment variable.
 - b. You have the JAVA_HOME environment variable setup and you have JAVA_HOME/bin in your PATH.
2. Edit the file *server-root/demo-ear/entity-ejb/DiaryLocalHome.java*. Replace 'public Local create' and 'public Local findByPrimaryKey' by 'public DiaryLocal create' and 'public DiaryLocal findByPrimaryKey'. This seems to be a bug in jrunwizard. There is no return type such as Local that is accessible from this location during compilation / execution.

3. Goto the location *server-root/demo-ear/entity-ejb* and execute 'javac demo/*.java'. If everything is fine, then the source files will be compiled and the class files will be placed inside the 'demo' directory itself.
4. That's it your bean is now ready and can be deployed.

Making the web application (JSP) talk to the bean

1. I am pasting the 2 files *index.jsp* and *web.xml* that you need to make changes to, to make your JSP page talk to your bean.

index.jsp

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="iso-8859-1" %>
<%@ page import="javax.ejb.*, javax.naming.*, demo.*" %>
<html>
  <head>
    <title>Connecting to Entity Bean: Diary</title>
  </head>

  <body>
    <h1>Connecting to Diary...</h1>
    <%
try {
  /* 1: Get the Context */
  Context ctx = new InitialContext();

  /* 2: Look up using the ENC (java:comp/env) */
  Object o = ctx.lookup("java:comp/env/ejb/Diary");
  /* Anything ahead of the java:comp/env part is referred from the web.xml's */
  /* Use a cast for a local bean */
  DiaryLocalHome home = (DiaryLocalHome) o;

  /* 3: Create specific EJBObject (insert records in the table)
     the create() method returns an object of type DiaryLocal
     Parameters in this method are actually the attributes of the table
     that is represented by the bean. */
  home.create(new Integer(1), "Mr. Incredible", "1234");
  home.create(new Integer(2), "Elastigirl", "1235");
  home.create(new Integer(3), "Dash", "1236");
  home.create(new Integer(4), "Violet", "1237");
  System.out.println("Created 4 records");

  /* 4: Search for every bean and then display the contents */
  DiaryLocal test = null;
  for (int i = 1; i <= 4; i++) {
    test = home.findByPrimaryKey(new Integer(i));
    out.print(test.getId() + " " + test.getName() + " " + test.getPhone());
    out.println("<br>");
  }
  System.out.println("Displayed 4 records");

  /* 5: Search and modify a bean (update a record in the table) */
  test = home.findByPrimaryKey(new Integer(4));
  test.setName("Purple");
  System.out.println("Modified 1 record");

  /* 6: Search and remove a bean (delete a record in the table) */
  test = home.findByPrimaryKey(new Integer(3));
  test.remove();
  System.out.println("Deleted 1 record");
}
catch (Exception e) {
```

```
    out.println("Something wrong...");
    e.printStackTrace();
}
%>
</body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <display-name>JRun Default Web Application</display-name>
  <description>The default web application for JRun</description>

  <error-page>
    <error-code>404</error-code>
    <location>/errorPages/404.jsp</location>
  </error-page>

  <!-- This is the only tag that was added to the original web.xml file -->
  <ejb-local-ref>
    <!-- The part of the lookup reference after java:comp/env/ -->
    <ejb-ref-name>ejb/Diary</ejb-ref-name>
    <!-- The type of the bean -->
    <ejb-ref-type>Entity</ejb-ref-type>
    <!-- Indicates the Local Home interface -->
    <local-home>demo.DiaryLocalHome</local-home>
    <!-- Indicates the Local Component interface -->
    <local>demo.DiaryLocal</local>
    <!-- The actual JNDI name of the EJB. This comes from the ejb-name tag in
server-root/demo-ear/entity-ejb/META-INF/ejb-jar.xml -->
    <ejb-link>Diary</ejb-link>
  </ejb-local-ref>

  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

</web-app>
```

2. You must read through this code and the included comments to understand what is happening.

Running the servlet

1. Restart the j2eeSamples server.
2. Point your browser to `http://machine-name:http-port-number-of-j2eeSamples/entity/` (eg. `http://pollux.cse.buffalo.edu:12345/entity/`)
3. This would execute your JSP once only. You cannot execute the JSP again without getting exceptions, since it would cause duplicate entries in the database. If you want to run it again, you must modify the JSP file to delete previous records first.