Developing an EJB3 Application

on WebSphere 6.1

using RAD 7.5

Introduction

This tutorial introduces how to create a simple EJB 3 application using Rational Application Developer 7.5 (RAD7.5 for short) and publish the application on a WebSphere Application Server 6.1 (WAS6.1 for short). In this tutorial we use a WindowsXP copy, on that have been installed WAS6.1 with EJB3 Feature pack and RAD7.5.

This tutorial is divided into 2 parts
Part 1 Configurating RAD7.5
Part 2 Developing an application

Part 1 Configuration RAD7.5

Application Developer is an Eclipse-based design and development toolset. It provides workspace management with its perspectives, editors, views, and plug-ins. If you have worked with Eclipse before, you should find it easy to design and develop enterprise applications with Application Developer. Start it with:

Start → Programme → IBM Software Delivery Platform → IBM Rational Application Developer 7.5 → IBM Rational Application Developer.
Step 1  Change the Perspectives to Java EE

You can open the perspective by selecting **Window → Open Perspective**, or by clicking the **Open Perspective** icon.

Step 2  Creating a server

One of the important views pinned to the Java EE perspective is the **Servers** View. It is the view of your configured servers. Right-click anywhere in the Servers view and select **new → Server**.

Select **WebSphere Application v6.1 Server** and click **Add** to create our Server runtime environment.
Navigate to the WAS6.1 installation directory, in this sample, c:\IBM\WebSphere\AppServer, click ok and 2 times next.
Because of the security setting of the server, we must enter the User ID and Password, in this sample, they both are admin, click Finish.

Now there is a new Server entry in the Servers View.

**Step 3  Creating a Database connection**

Another important view of java EE perspective is the Data Source Explorer. It should locate in the same bar where the Servers Views locates.
An EJB3 Application

If the view does not appear, click Window->Show View-> Data Source Explorer to open it.

Click right button on Databases and select New

Now select a DB2 for z/OS database manager.

In this tutorial we use DB2 on z/OS, the database name is S1D931. Type S1D931 as Location, 134.2.205.54 as Host, 4019 for Port. Enter your own username and password that you get from the Administrator. For example, username: prak435 password: 1234
Now a new database connection named S1D931 is created and appears in the view. After expanding the connection you will find too many schemas, because this database is used for many users. Click right button on this connection and click Properties. In the next Window choose Default Schema Filter on the left side, deselect Disable filter and enter your own username (case sensitive), for example PRAK435, click ok. Click F5 to refresh the connection view, there is only your schema left.
Step 4 Creating a table and inserting data

Click right button on the S1D931, select **New SQL Script**
In the text area enter following code to create a new table and insert some data.

```
CREATE TABLE ITEM (  
    ID INTEGER NOT NULL PRIMARY KEY,  
    NAME VARCHAR(20) WITH DEFAULT NULL,  
    QUANTITY INTEGER WITH DEFAULT NULL  
)
```

Click right button anywhere in the text area and select **Run SQL**
If the table is successfully created, the **SQL Results View** will show the following message.
An EJB3 Application

Now we insert some data into the table, enter the following code in the text area and execute them.

```sql
INSERT INTO ITEM VALUES (1001, 'Book', 20);
INSERT INTO ITEM VALUES (1002, 'CD', 20);
INSERT INTO ITEM VALUES (1003, 'DVD', 20)
```

The SQL Results View will show you whether the SQL codes are successfully executed.

### Part 2 Creating an Enterprise Application

EJB3 enhances the EJB2.X and makes a great progress. In EJB2.X all resources contact each other through JNDI, EJB3 simplifies the model, the most resources can be defined using a tag `@`.

A typical EJB3 project consists of 4 parts.

- **_JPA Project** persists the data stored in a database.
- **_Client Project** defines the business logic that can be accessed by remote users.
- **_EJB Project** implements the interfaces defined in the Client project.
- **_Dynamic Web Project**
Step 1 Creating an Enterprise Application Project

Select File → New → Project.
In the New Project wizard, select Java EE → Enterprise Application Project.

In the EAR Application project panel, type Shop as the Project name. For Target Runtime, select WebSphere Application Server v6.1. For EAR version, select 5.0 and select Default Configuration for WebSphere Application Server v6.1 as Configuration.

---

An EJB3 Application

---
Click **Next**. Do not select any existing modules (if there are any). Select **Generate Deployment Descriptor**.

Click **Finish**. If you are not already in the Java EE perspective, you are prompted to switch to it. Click **Yes**
Do not worry about the error (the Red Cross icon in the Shop project). It is because the deployment descriptor file, application.xml, does not contain any modules.

**Step 2 Creating a JPA project**

Select File → New → Project, then select JPA → JPA Project. Click Next. Enter ShopJPA as Project name, select Add project to an EAR, and select Shop as EAR Project Name. Notice the Configuration: Utility JPA project with Java 5.0. Click Next.

The JPA Facet panel is the most important panel in the JPA Project wizard. We specify how a JPA provider manages the entities and whether they should be listed in the persistence.xml file or not. You can optionally create an orm.xml file, which is used for entity-relational mapping. For Connection select “S1D931” which is a DB2
connection we created in part 1. Click **Finish**

Creating a JPA entity
Click right button on **ShopJPA** project then select **JPA Tools**->**Generate Entities**...

Select “S1D931” as **Connection** and “PRAK435” (you should select your schema) as **Schema**. Click **Next**
Type shop as **Package** and select Item shown in Tables. Click Finish

Now a entity bean is automatic created without any Typing. The source code is like the below text

```java
package shop;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
/* define an Entity Bean */
public class Item implements Serializable {
    @Id /* define the key for the bean */
```
An EJB3 Application

private int id;

private String name;

private int quantity;

private static final long serialVersionUID = 1L;

public Item() { super(); }

public int getId() { return this.id; }

public void setId(int id) { this.id = id; }

public String getName() { return this.name; }

public void setName(String name) { this.name = name; }

public int getQuantity() { return this.quantity; }

public void setQuantity(int quantity) { this.quantity = quantity; }

}

Then we add 2 Queries, one without parameter, another with a parameter „id“.
Click right button on Item shown in Project Explorer, select JPA Tools->Configure JPA Entities to open the Configuration window. Select Item from the table list and click next.

Select Named Queries and click Add
Select all of the three Attributes, type „getItem“ as **Named Query Name**, type „select i from Item i“ in the text area of **Query Statement Panel**. Click **OK**
Click **Add** once again. Select all of the three Attributes, type „get(itemId)“ as **Named Query Name**, type „SELECT i FROM Item i where i.id =:id“ in the text area of **Query Statement Panel**. Click **OK**

Click **Finish**
Open the source code of Item.java, you will find the source code is changed. There are a few new lines to the code that are shown below in green.

```
......
@Entity
@NamedQuery(   
    @NamedQuery(name="getItem", query = "SELECT i FROM Item i"),
    @NamedQuery(name="getItemByID", query = "SELECT i FROM Item iparameter
```
WHERE i.id = :id")
})
public class Item implements Serializable {
......

Mapping runtime environment
Double click persistence.xml in ShopJPA project to edit it.
Select JTA as Transaction Type, for both of JTA Data Source and Non JTA Data Source enter „jdbc/db2“, which is the Data Source we defined in WAS6.1. About how to define a data source in WAS6.1, please see tutorial „Installation and Configuration of WAS6.1“.

Expand Persistence Unit → Properties and add a new Property named openjpa.jdbc.Schema to the value "PRAK435".
An EJB3 Application

Click the **Source** tab to see the content of persistence.xml, it's shown below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="ShopJPA">
    <jta-data-source>jdbc/leia</jta-data-source>
    <non-jta-data-source>jdbc/leia</non-jta-data-source>
    <class>
      shop.Item
    </class>
    <properties>
      <property name="open.jdbc.Schema" value="PRAK435"/>
    </properties>
  </persistence-unit>
</persistence>
```

Save the Changes.

Also add the class shop.Item (as shown in the code above) into the source or add it via the GUI browser.
Step 3 Creating an EJB 3.0 project for the session bean

In the Java EE perspective, select File → New → Project and then EJB → EJB Project. Click Next.
Enter ShopEJB as the project name, select Default Configuration for WebSphere Application Server v6.1, and add the project to the Shop enterprise application, click Next.

Select **Create an EJB Client JAR module** to hold the client interfaces and classes, and accept the default names. Select **Create an EJB Client JAR module to hold the client interfaces and classes**, and accept the default names.
The EJB 3.0 project is accessed by Web clients that are developed with Application Developer as Dynamic Web Projects. The ShopEJBClient.jar contains public (business) interfaces of the EJBs, and as such, is the only jar file needed by clients (such as Web clients). The EJB Client project is therefore a dependency of the client projects that access the EJB 3.0 bean.
Besides, it is always a good design approach to separate the public interface (the business interface of the EJB 3.0 bean) from its implementation (the implementation class of the EJB 3.0 bean). Having two projects with a well-established purpose makes their maintenance easy.
Click Finish

Adding the persistence module to the EJB module
The session bean for the shipping cart will access the Item entity bean. We add the JPA module as a dependency to the EJB client module:
Click right button on the ShopEJBClient project and select Properties. Select Java EE Module Dependencies. Select ShopJPA.jar and click OK.
Creating a business interface
An EJB 3.0 session bean implements a business interface. We create this interface before we create the session bean.

Create a business interface `shop.Cart` in the `ShopEJBClient` project (under ejbModule).

The interface has four public methods
An EJB3 Application

- **public void add(int id)** — Add an item to the cart.
- **public void remove(int id)** — Remove an item from a cart.
- **public List<Item> getItems()** — List the available items in the store.
- **public List<Item> checkout()** — Check out with the selected items.

The code is displayed below:

```java
package shop;
import java.util.List;
public interface Cart {
    public List<Item> getItems();
    public List<Item> checkout();
    public void add(int id);
    public void remove(int id);
}
```

**Creating a stateful session bean**

Having the business interface `shop.Cart` defined, we create its implementation as a stateful session bean in the `ShopEJB` project. The stateful session bean can handle entities in sync with the database.

Add ShopEJBClient and ShopJPA projects as dependencies of the ShopEJB project. Use the **Java EE Module Dependencies** in the Properties dialog for `ShopEJB`. Click **OK**.

In the ShopEJB project, create the bean class `shop.CartBean`, which implements the `shop.Cart` interface. The code is displayed below:

```java
package shop;
import java.util.ArrayList;
import java.util.List;
import javax.ejb.Remove;
import javax.ejb.Stateful;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.PersistenceContextType;
import javax.persistence.Query;
@Stateful
public class CartBean implements Cart {
    @PersistenceContext(type = PersistenceContextType.EXTENDED)
    EntityManager em;
    private List<Item> selectedItems = new ArrayList<Item>();
    public void add(int id) {
        Query query = em.createNamedQuery("getItemByID");
        query.setParameter("id", id);
        Item item = (Item) query.getSingleResult();
    }
}
```
item.setQuantity(item.getQuantity()-1);
   selectedItems.add(item);
}

@Remove
public List<Item> checkout() {
   System.out.println("Checkout:");
   Item[] items = selectedItems.toArray(new Item[selectedItems.size()]);
   for (int i=0; i< items.length; i++){
      Item item = items[i];
      System.out.println(item.getId() + " " + item.getName());
   }
   em.flush();
   return selectedItems;
}

public List<Item> getItems() {
   return em.createNamedQuery("getItem").getResultList();
}

public void remove(int id) {
   Query query = em.createNamedQuery("getItemByID");
   query.setParameter("id", id);
   Item item = (Item) query.getSingleResult();
   item.setQuantity(item.getQuantity()+1);
   selectedItems.remove(item);
}

The stateful session bean uses an extended persistence context to keep the entities attached to the entity manager. The persistence context (the set of entities) is maintained between transactions rather than being discarded at the end of each transaction. The changes are written to the database at the end of every transaction, in this case are at the end of each add or remove method. When executing the method annotated with @Remove, the stateful bean is removed.

In methods add and remove we use the named query "getItemByID", which is defined in Item entity bean.

Open the Properties and specify the ShopJPA and ShopEJBClient projects as its Java EE Module Dependencies

Step 4 Creating an Dynamic Web Project

Click Project->New->Dynamic Web Project. Type “ShopWAR” as Project name, choose “Default Configuration for WebSphere Application Server v.6.1” for
Configuration, select Add project to an EAR. Click Finish

Open the Properties and specify the ShopJPA and ShopEJBClient projects as its Java EE Module Dependencies

Create a servlet named shop.ShopServlet by clicking right button on the Deployment Descriptor and New →Servlet
Click Finish and the ShopServlet opens in the editor.

Type the following code for the class

```java
package shop;
import java.io.IOException;
import java.util.List;
import javax.ejb.EJB;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class ShopServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public static final String CART_REF_NAME = "shop.cart";

    public ShopServlet() {
    }
```
super();
}

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    HttpSession session = request.getSession();
    Cart cart = (Cart) session.getAttribute(CART_REF_NAME);
    if (cart == null) {
        try {
            System.err.println("runs here");
            Context ctx = new InitialContext();
            cart = (Cart) ctx.lookup("java:comp/env/ejb/Cart");
            session.setAttribute(CART_REF_NAME, cart);
        } catch (NamingException ne) {
            throw new ServletException(ne);
        }
    } catch (NamingException ne) {
        throw new ServletException(ne);
    }
    //**** the business logic ****/
    java.io.PrintWriter out = response.getWriter();
    // display the available items
    out.println("<h2>Items</h2>");
    List<Item> itemlist = cart.getItems();
    Item[] items = itemlist.toArray(new Item[itemlist.size()]);
    for (int i=0; i< items.length; i++) {
        Item item = items[i];
        out.println(item.getId() + " " + item.getQuantity() + " " + item.getName() + "<br>");
    }
    // fill the shopping cart
    cart.add(1001); // add a book to cart
    cart.add(1001); // add a book to cart
    cart.add(1003); // add a dvd to cart
    cart.add(1002); // add a cd to cart
    cart.add(1002); // add a cd to cart
    cart.add(1002); // add a cd to cart
    cart.remove(1001); // remove a book from cart
    cart.remove(1002); // remove a cd from cart
    // display the remaining items
    out.println("<h2>Remaining Items</h2>");
    itemlist = cart.getItems();
    items = itemlist.toArray(new Item[itemlist.size()]);
for (int i=0; i< items.length; i++){
    Item item = items[i];
    out.println(item.getId() + " &nbsp; " + item.getQuantity()
            + " &nbsp; " + item.getName()+ "<br>");
}
// checkout and list the content of the shopping cart
out.println("<h2>Checkout Cart</h2>");
itemlist = cart.checkout();
items = itemlist.toArray(new Item[itemlist.size()]);
for (int i=0; i< items.length; i++){
    Item item = items[i];
    out.println(item.getId() + " &nbsp; " + item.getName()+ "<br>");
}
// remove session data
session.removeAttribute(CART_REF_NAME);
}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    doGet(request, response);
}

Mapping the EJB logical reference
Open the Deployment Descriptor editor for ShopWAR
Switch to Source tab of the Web Deployment Descriptor editor and add the *ejb/Cart* reference. Because the Web application and the enterprise bean are in the same enterprise application, we can use *ejb-local-ref* element to create the reference

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"......>
<display-name>ShopWAR</display-name>
<servlet>
<description></description>
<display-name>ShopServlet</display-name>
<servlet-name>ShopServlet</servlet-name>
<servlet-class>shop.ShopServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>ShopServlet</servlet-name>
<url-pattern>/ShopServlet</url-pattern>
</servlet-mapping>
<ejb-local-ref>
<description>
```
An EJB3 Application

```xml
<description>
  <ejb-ref-name>ejb/Cart</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home/></local-home>
  <local>shop.Cart</local>
  <ejb-link>CartBean</ejb-link>
</ejb-local-ref>
<welcome-file-list>
  ...
</web-app>
```

**Step 5 Running project on Server**

Click right button on **Shop** project, choose **Run As-> Run on Server**, choose the server environment that we created in part 1.
An EJB3 Application

Click Finish
Waiting until all projects are started.

Open a Web browser, type “localhost:9080/ShopWAR/ShopServlet”, you will get the page like the figure shown below.
An EJB3 Application

Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>15</td>
</tr>
<tr>
<td>1002</td>
<td>16</td>
</tr>
<tr>
<td>1003</td>
<td>18</td>
</tr>
</tbody>
</table>

Remaining Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>14</td>
</tr>
<tr>
<td>1002</td>
<td>14</td>
</tr>
<tr>
<td>1003</td>
<td>17</td>
</tr>
</tbody>
</table>

Checkout Cart

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Book</td>
</tr>
<tr>
<td>1003</td>
<td>DVD</td>
</tr>
<tr>
<td>1002</td>
<td>CD</td>
</tr>
<tr>
<td>1002</td>
<td>CD</td>
</tr>
</tbody>
</table>