Dali Object-Relational Mapping Tool

Advanced Tutorial

Release 1.0.0

June 2007

This tutorial provides information on building and deploying dynamic Web application for an order-entry system using Dali. The mapped classes are packaged into a Web application, demonstrating one way to deploy JPA applications. This tutorial focuses on using Dali to map your classes to a relational database – the details of the Web application are not discussed in this document.

This tutorial includes:

- Requirements and installation
- Dali Advanced Tutorial

For additional information, please visit the Dali home page at:
`http://www.eclipse.org/webtools/dali/main.php`.

1 Requirements and installation

Before building this tutorial, ensure that your environment meets the following requirements:

- Eclipse 3.3 (`http://www.eclipse.org/downloads`)
- Java Runtime Environment (JRE) 1.5 (`http://java.com`)
- Eclipse Web Tools Platform (WTP) 2.0 (`http://www.eclipse.org/webtools`)
- Java Persistence API (JPA) for Java EE 5. This tutorial uses the TopLink Essentials JPA implementation that can be obtained from:

  `http://otn.oracle.com/jpa`

- Java Server Faces (JSF) 1.1 for Java EE5. The reference implementation can be obtained from:

  `https://javaserverfaces.dev.java.net/`

  Note: This tutorial requires JSF 1.1. Do not use JSF 1.2.

- J2SE Application server. This tutorial uses Apache Tomcat 5.5 that can be obtained from:

  `http://tomcat.apache.org/`

- Relational database. This tutorial uses Apache Derby 10.1.3.1 that can be obtained from:

  `http://db.apache.org/derby/`

- Tutorial source files (including the nonpersistent files, implementation classes, and Web content):
  `http://www.eclipse.org/webtools/dali/docs/tutorial/jsf/Dali_Tutorial_`

Application.zip. Unzip these files to your <DALI_TUTORIAL_HOME>. The following table identifies the files in the dali_tutorial_application.zip:

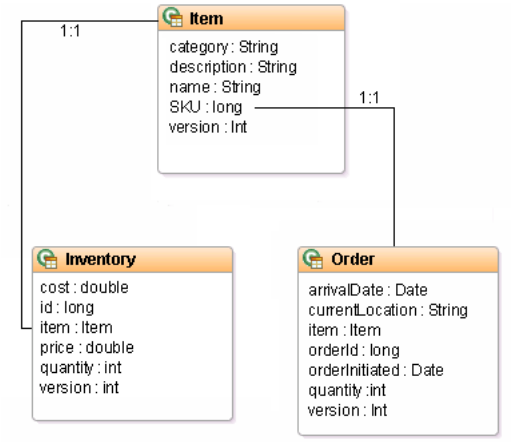
File	Description
commons-cli-1.0.jar	Used to parse the command line arguments when populating the database.
Dali_Tutorial_Web.zip	Java source and Web files used for the tutorial dynamic Web project.
dalimodel.jar	The completed model project. This is used to create and populate the database and is also included for reference.
populatedb.bat	Script to create and populate the tutorial database schema.
populatedb.jar	Java source files used to create and populate the database.
sources.zip	Java source for the domain model classes that will be used in this tutorial.

Refer to http://www.eclipse.org/webtools/dali/gettingstarted_main.html for additional installation information.

2 Dali Advanced Tutorial

In this tutorial, you will create a dynamic Web application for an order-entry system. Figure 1 illustrates the object model for this tutorial.

Figure 1 Advanced Tutorial Object Model



The **Item** class represents the items that can be ordered or maintained in inventory. The **Inventory** class models items that are in inventory. The **Order** class represents a request for delivery of a particular item.

2.1 Generate the tutorial database schema

The advanced tutorial project uses three database tables to store each order: INVENTORY, ITEM, and ORDER_TABLE.

Table 1 Tutorial Database Schema

Table	Column	Type	Details
INVENTORY	COST	NUMBER(10,4)	
	ITEM_SKU	NUMBER(10,0)	Primary Key, references ITEM.SKU
	PRICE	NUMBER(10,4)	
	QUANTITY	NUMBER(10,0)	
	VERSION	NUMBER(10,0)	
ITEM	CATEGORY	VARCHAR2(255)	
	DESCRIPTION	VARCHAR2(255)	
	NAME	VARCHAR2(255)	
	SKU	NUMBER(10,0)	Primary Key
	VERSION	NUMBER(10,0)	
ORDER_TABLE	ARRIVALDATE	DATE	
	CURRENTLOCATION	VARCHAR2(255)	
	ITEM_SKU	NUMBER(10,0)	Foreign Key, references ITEM.SKU
	ORDERID	NUMBER(10,0)	Primary Key
	ORDERINITIATED	DATE	
	QUANTITY	NUMBER(10,0)	
	VERSION	NUMBER(10,0)	

Included in the tutorial source file (`http://www.eclipse.org/webtools/dali/docs/tutorial/jsf/Dali_Tutorial_Application.zip`) are the scripts that will create and populate the tutorial database. By default, the scripts will use the following login information:

- `driver=org.apache.derby.jdbc.ClientDriver`
- `url=jdbc:derby://localhost:1527/sample;create=true`
- `user=dali`
- `password=dali`

1. Install Apache Derby and start the Derby database, using the Network Server framework.

   Refer to the Derby documentation (`http://db.apache.org/derby/`) for details. Be sure to correctly set your DERBY_INSTALL and classpath variables.

2. Place the `toplink-essentials.jar` and `derbyclient.jar` files in the same directory as the `populatedb.bat` file.

3. Execute the `populatedb.bat` file to create and populate the tutorial database. The script executes the following command:

   ```
   java -classpath
   populatedb.jar;dalimodel.jar;commons-cli-1.0.jar;toplink-essentials.jar;derbyclient.jar
   oracle.toplink.jpa.example.inventory.util.PopulateDatabase
   ```

To override the default login information, include your JDBC driver JAR and login information. For example:

```
java -classpath
createschema.jar;dalimodel.jar;commons-cli-1.0.jar;toplink-essentials.jar;ojdbc14.jar
oracle.toplink.jpa.example.inventory.util.PopulateDatabase -user scott -password tiger -driver
oracle.jdbc.OracleDriver -url jdbc:oracle:thin:@localhost:1521:ORCL
```

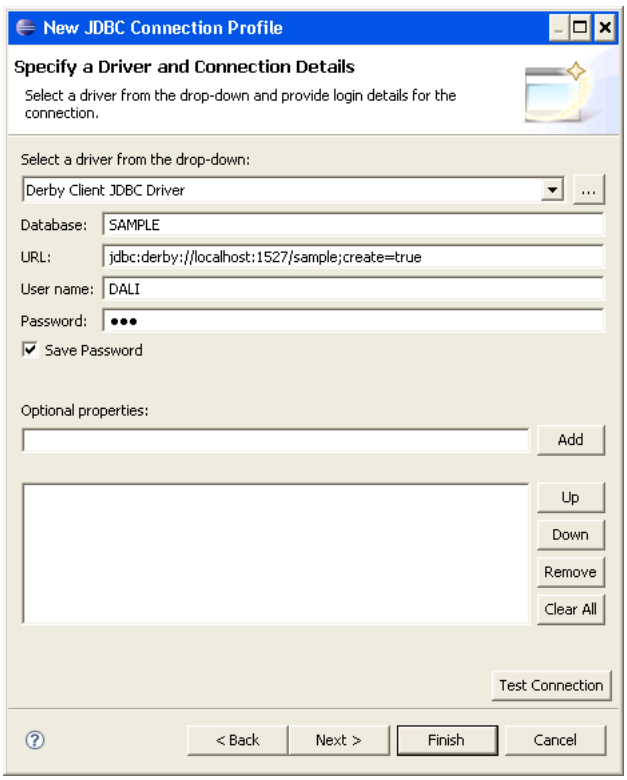
2.1.1 Create a database connection

After creating and populating the database you will need to create a database connection to use with the tutorial application. An active database connection is required to complete the tutorial application.

1. In Eclipse, use the New Connection wizard to create a database connection. Refer to "Creating a Connection Profile" in the Eclispe online help for more information.

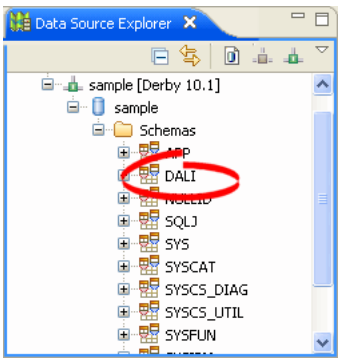
> **Note:** Use the SQL Model-JDBC Connection profile to create the connection.

Figure 2 Creating a Database Connection



2. Open the Data Source Explorer view to display the tutorial database.

Figure 3 Database Explorer



2.2 Create a Java project

To begin the tutorial, you must create a new Eclipse project. This Java project will contain the model classes for the tutorial application.

1. Select **File > New > Project**. The New Project dialog appears.
2. On the New Project dialog, select **JPA > JPA Project** and click **Next**. The New JPA Project wizard appears.
3. On the New JPA Project page, enter the following information and click **Next**.
   - In the **Project name** field enter `Dali_Tutorial_Model`.
   - In the **Target Runtime** area, select **Apache Tomcat**.

     If you do not have a defined Apache Tomcat target runtime, you must create one. Refer to "Defining the installed server runtime environments" in Web Application Development User Guide for details.
   - In the **Configurations** area, select **Utility JPA Project with Java 5.0**.

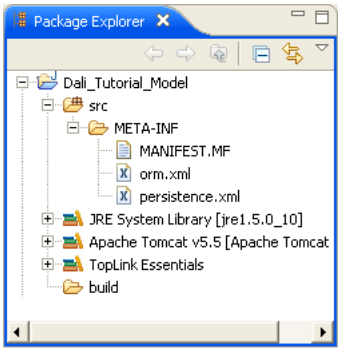
   The Project Facets dialog appears.
4. On the Project Facets page, select the following options and click **Next**.
   - **Java Persistence 1.0**
   - **Java 5.0**

   The JPA Facet dialog appears.
5. On the JPA Facet dialog, enter the following information:
   - In the **Platform** field, select `Generic`.
   - In the **Connection** field, select the database connection that you created previously.
6. In the JPA Implementation Library area, click **Configure default JPA implementation**. The JPA Preferences dialog appears.
7. Click **Configure user libraries**. The User Libraries dialog appears.
8. Click New to create a new user library, named **TopLink Essentials**, that contains the `toplink-essentials.jar` (see "Requirements and installation" on page 1 for details).
9. On the User Libraries dialog, click **OK**. The JPA Facet dialog appears.

10. On the JPA Facet page, in the **JPA implementation** area select **Use implementation library** then use the drop-list to select the `TopLink Essentials` implementation library that you previously created, and click **Finish**.
11. Complete the remaining fields on the JPA Facet page and click Finish.
    - In the **Persistent class management** area, select `Annotated classes must be listed in persistence.xml`.

Eclipse creates the JPA project and opens the JPA perspective.

Figure 4 New JPA Project



2.3 Create Java classes

The Advanced Tutorial Object Model contains three entities: **Inventory**, **Item**, and **Order**. Use this procedure to add the classes to the project. Later, you will change them to persistent entities.

1. Right-click the project in the Package Explorer and select **New > Class**. The New Java Class dialog appears.
2. On the New Java Class page, enter a package name and class name and click **Finish**.

   For this tutorial, use `org.eclipse.dali.example.jsf.inventory.model` as package and `Inventory` as the class name.

   Eclipse adds the **Inventory** class to the Package Explorer.

Repeat this procedure to add the **Item** and **Order** classes.

2.3.1 Build the entities

Before mapping the entities to the database, you must add the necessary fields to each entity.

1. Add the following fields to the **Inventory** entity:

```
protected double cost;
private long id;
protected Item item;
protected double price;
protected int quantity;
protected int version;
```

   Add `get()` and `set()` methods for the following:

   - cost

- item
- price
- quantity

2. Add the following fields to the **Item** entity:

```
protected String category;
protected String description;
protected String name;
protected long sKU;
protected int version;
```

Add `get()` and `set()` methods for the following:

- category
- description
- name
- SKU
- version

3. Add the following fields to the **Order** entity:

```
protected Date arrivalDate;
protected String currentLocation;
protected Item item;
protected long orderId;
protected Date orderInitiated;
protected int quantity;
protected int version;
```

Import `java.util.Date`.

Add `get()` and `set()` methods for the following:

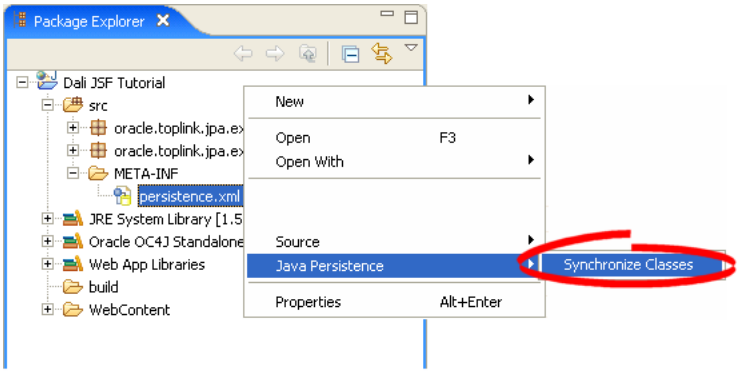
- arrivalDate
- currentLocation
- item
- orderId
- orderInitiated
- quantity

2.3.2 Create persistent entities and associate with a database table

Now you will change each class to a persistent entity. You must also associate each entity with its primary database table.

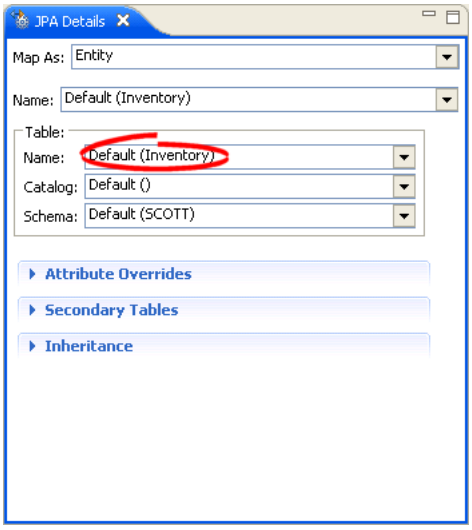
1. In the Package Explorer view, open **Inventory.java**.
2. In the JPA Structure view, select the **Inventory** entity.
3. In the JPA Details view, in the Map As field select **Entity**. Eclipse adds the `@Entity` annotation to the class.
4. In the Package Explorer view, right-click the **persistence.xml** file select **JPA Tools > Synchronize Classes**. This will update the `persistence.xml` file with the newly added entity.

Figure 5 Synchronizing the persistence.xml File



In the JPA Structure Properties view, notice that Dali has automatically identified the default table, **Inventory**, associated with the entity.

Figure 6 JPA Details view for the Inventory Entity



Repeat this procedure to create entities from the **Item** and **Order** classes and then associate the entities with the database tables. By default, entities are associated with a similarly named database table and Dali identifies these defaults. Like the Inventory entity, even though you have not explicitly associated the **Item** entity with a database table, there is no error in the Problems view because the entity name, Item, is identical to the table name (Item).

Because the **Order** entity is named differently than the database table (ORDER_TABLE), you must explicitly create the association (as shown in Figure 6). Dali adds the `@Table(name="ORDER_TABLE")` annotation to the Order entity.

Remember to resynchronize the `persistence.xml` file after creating and associating the entities.

Note: Depending you your specific database type, you may need to select the database schema and table information

---

2.4 Create OR mappings

Now you're ready to map the attributes of each persistent entity to columns in the appropriate database table. For the tutorial application, you will use the following mapping types:

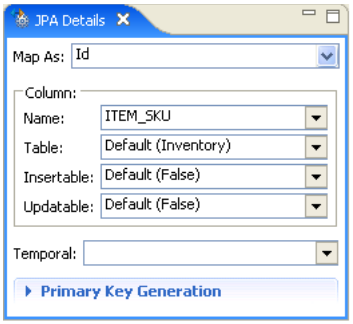
- ID mappings
- Basic mappings
- One-to-one mappings
- One-to-many mappings
- Version mappings

2.4.1 Create ID mappings

Use an **ID Mapping** to specify the primary key of an entity. Each persistent entity must have an ID. Notice that the Problems view reports that each entity "does not have Id or EmbeddedId."

1. In the Package Explorer view, open **Inventory.java**.
2. Expand the **Inventory** entity in the JPA Structure view and select the **id** field. The JPA Details view displays the properties for the field.
3. In the Map As field, select **ID**.

Figure 7 ID Mapping for id Field

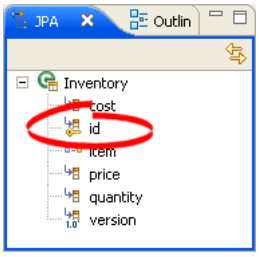


4. Use this table to complete the remaining fields on the General tab in the Persistence Properties view.

Property	Description
Map As	Defines this mapping as an **ID Mapping**. Dali adds the `@Id` annotation to the entity.
Column	The database column for the primary key of the table associated with the entity. Select **ITEM_SKU**. Because the database column (ITEM_SKU) is named differently than the entity field (id), Dali adds the `@Column(name="ITEM_SKU")` annotation. Anytime you override the default, Dali adds the `@Column` annotation.
Insertable	The value of the **id** field is obtained from the Item entity. Select **False**. Dali adds `insertable = false` to the annotation.
Updatable	The value of the **id** field is updated from the Item entity. Select **False**. Dali adds `updatable = false` to the annotation.

In the JPA Details view, the **id** field is identified as the primary key by the following icon:

Figure 8 JPA Structure for Inventory Entity



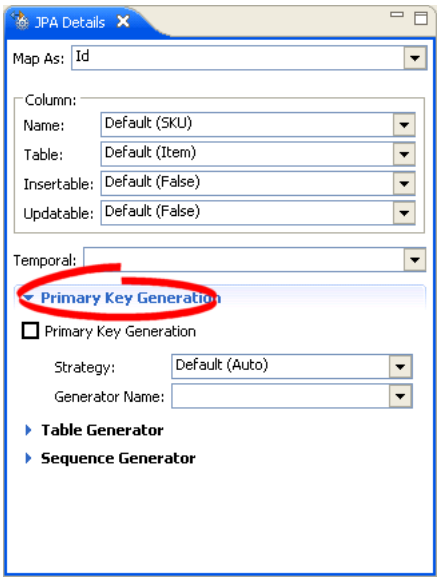
Repeat this procedure to map the following primary keys (as shown in Table 1, " Tutorial Database Schema"):

- The **SKU** field of the **Item** entity to the SKU column of the ITEM table.
- The **orderId** field of the **Order** entity to the ORDERID column of the ORDER_TABLE table.

For both of these mappings:

1. Expand the **Primary Key Generation** area.

Figure 9 Primary Key Generation for id Field



2. Select the **Primary Key Generation** option.
3. Use this table to complete the remaining fields in the JPA Details view.

Property	Description
Generated Value	These fields define how the primary key is generated. Dali adds the `@GeneratedValue` annotation to the entity.
Strategy	For the tutorial project, leave this as the Default (**Auto**).
Generator Name	Leave this field blank.

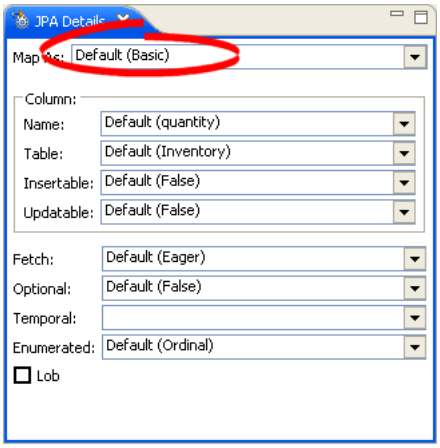
2.4.2 Create basic mappings

Use a **Basic Mapping** to map an attribute directly to a database column. In the object model, the **quantity** field of the **Inventory** class maps directly to the QUANTITY column of the INVENTORY database table.

1. In the Package Explorer view, open **Inventory.java**.
2. In the JPA Structure view, select the **quantity** field of the **Inventory** entity. The JPA Details view displays the properties for the field.

   Notice that Dali has already identified the mapping as the Basic mapping type. By default, all attributes of the following types use Basic mapping: Java primitive types, wrappers of the primitive types, `java.lang.String`, `java.math.BigInteger`, `java.math.BigDecimal`, `java.util.Date`, `java.util.Calendar`, `java.sql.Date`, `java.sql.Time`, `java.sql.Timestamp`, `byte[]`, `Byte[]`, `char[]`, and `Character[]`.

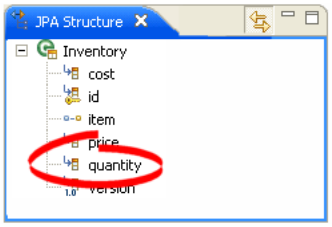
Figure 10 Basic Mapping for quantity



Notice that Dali has automatically identified the QUANTITY field and INVENTORY table for this mapping. Dali identifies the defaults for the mapping.

In the JPA Details view, the **quantity** field is identified as a basic mapping as shown in the following figure:

Figure 11 JPA Details view for Inventory Entity



Repeat this procedure to review each of the following **Basic** mappings:

- **Item** entity
  - **description** field to DESCRIPTION column
  - **name** field to NAME column
  - **category** field to CATEGORY column
- **Order** Entity

- **orderInitiated** field to ORDERINITIATED column

  In the JPA Details view, use the Temporal field to select **Date**. Dali adds the `@Temporal(DATE)` annotation to the mapping.

- **arrivalDate** field to ARRIVALDATE column

  In the JPA Details view, use the Temporal field to select **Date**. Dali adds the `@Temporal(DATE)` annotation to the mapping.

- **currentLocation** field to CUURENTLOCATION column
- **quantity** field to QUANTITY column

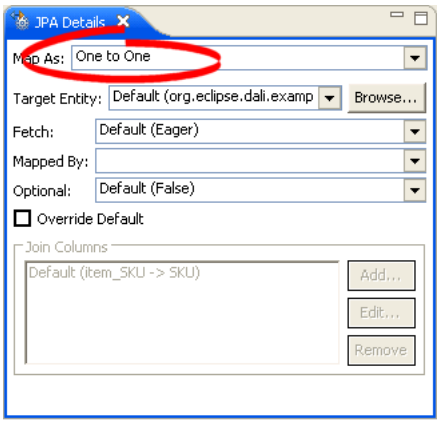
- **Inventory** Entity
  - **cost** field to COST column
  - **price** field to PRICE column

2.4.3 Create one-to-one mappings

Use a **One-to-One Mapping** to define a relationship from an attribute to another class, with one-to-one multiplicity to a database column. In the object model, the **item** field of the **Inventory** class has a one-to-one relationship to the **Item** class; each inventory object contains a single item.

1. In the Package Explorer view, open **Inventory.java**.
2. In the JPA Structure view, select the **item** field of the **Inventory** entity. The JPA Details view displays the properties for the field.
3. In the Map As field, select **One-to-One**.

Figure 12 One-to-one Mapping for item

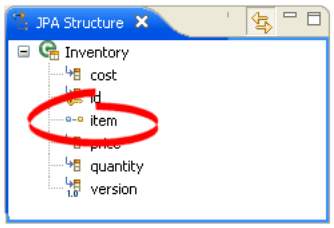


4. Dali has identified the default Target Entity for the mapping: `org.eclipse.dali.example.jsf.inventory.model.Item`.

   Leave the other fields with their default values.

In the JPA Structure view, the **item** field is identified as a one-to-one mapping, as shown in the following figure:

Figure 13 JPA Structure view for Inventory Entity



Repeat this procedure to create the following additional one-to-one mapping:

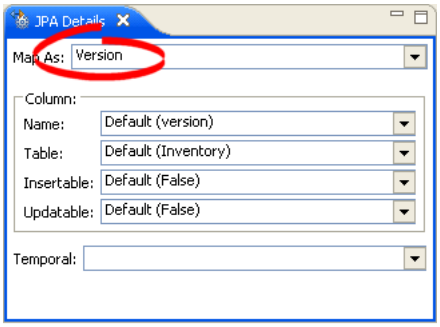
- Create a one-to-one mapping from the **item** attribute of the **Order** entity to the **Item**.

2.4.4 Create version mappings

Use a **Version Mapping** to specify the database field used by a persistent entity for optimistic locking.

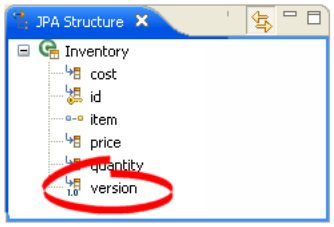
1. In the Package Explorer view, open **Inventory.java**.
2. In the JPA Structure view, select the **version** field of the **Inventory** entity. The JPA Details view displays the properties for the field.
3. In the Map As field, select **Version**.

Figure 14 Version Mapping for version



Notice that Dali has identified the default column in the INVENTORY database table. In the JPA Details view, the **Version** field is identified as a version mapping, as shown in the following figure:

Figure 15 JPA Details view for Inventory Entity



Repeat this procedure to create following version mappings:

- **version** attribute of the **Order** entity
- **version** attribute of the **Item** entity

2.5 Add the queries

You must define the following named queries in the **Inventory** and **Order** entities:

- `inventoryForCategory`
- `shippedOrdersForItem`
- `pendingOrdersForItem`

The tutorial source files include the code that will execute the queries at runtime, so they need to be defined:

1. Add the following query to the **Inventory** entity, immediately following the `@Entity` annotation.

```
import javax.persistence.NamedQuery;

@NamedQuery(name="inventoryForCategory", query="SELECT i FROM Inventory i WHERE i.item.category =
:category and i.quantity <= :maxQuantity")
```

2. Add the following queries to the **Order** entity.

```
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;

@NamedQueries({
    @NamedQuery(name="shippedOrdersForItem", query="SELECT o FROM Order o JOIN o.item i WHERE
i.sKU = :itemId and o.arrivalDate is not null"),
    @NamedQuery(name="pendingOrdersForItem", query="SELECT o FROM Order o WHERE o.item.sKU =
:itemId and o.arrivalDate is null")
})
```

2.6 Manage the persistence.xml file

When you originally added persistence to the project on the Add Persistence dialog, you selected to create the `persistence.xml`. Dali created a basic file, containing the persistence unit and provider information.

Use this procedure to add the entities to the `persistence.xml` file.

1. If you have not synchronized the `persistence.xml` file, do so now. Right-click the `persistence.xml` file in the Package Explorer and select **JPA > Synchronize Classes**.

   Dali adds the necessary `<class>` elements to the `persistence.xml` file:

```
<class>org.eclipse.dali.example.jsf.inventory.model.Inventory</class>
<class>org.eclipse.dali.example.jsf.inventory.model.Item</class>
<class>org.eclipse.dali.example.jsf.inventory.model.Order</class>
```

2. Add your database-specific login information to the `persistence.xml` file, inside the `<persistence-unit>` element, after the final `<class>` element. For example:

```
<properties>
  <property name="toplink.logging.level" value="FINEST"/>
  <property name="toplink.jdbc.driver" value="org.apache.derby.jdbc.ClientDriver"/>
  <property name="toplink.jdbc.url" value="jdbc:derby://localhost:1527/sample;create=true"/>
  <property name="toplink.jdbc.user" value="dali"/>
  <property name="toplink.jdbc.password" value="dali"/>
</properties>
```

This information should be identical to the values that you used when creating the database connection (see Figure 2 on page 4).

3. Change the persistence unit name to **default**:

```
<persistence-unit name="default">
```

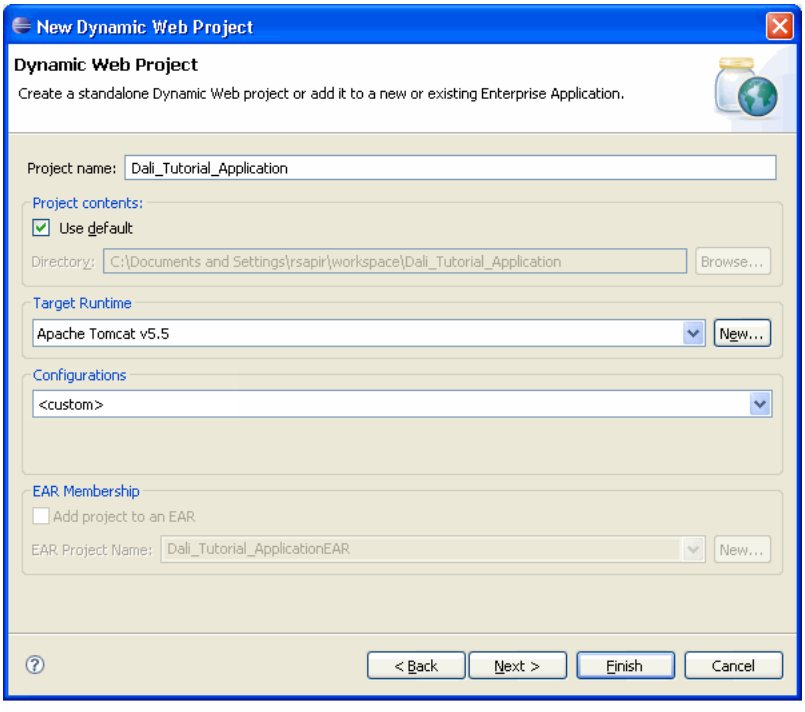
4. Save the `persistence.xml` file.

2.7 Create the dynamic Web project

To complete the tutorial, you must create a Web project. This Web project will contain the business logic and presentation files for the tutorial application.

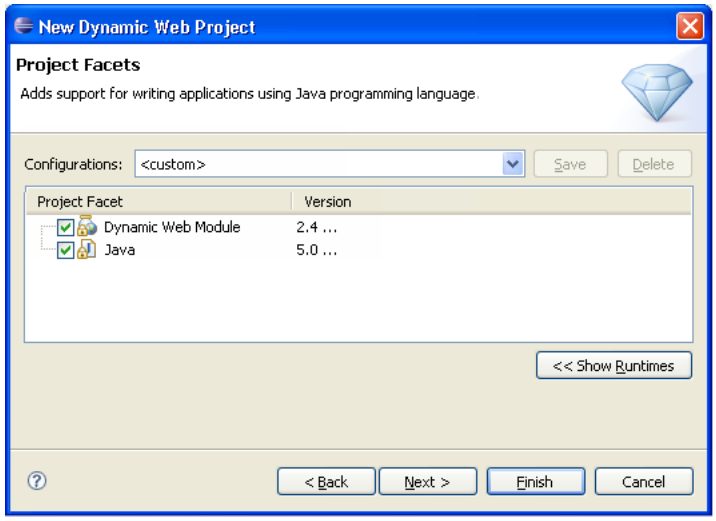
1. Select **File > New > Project**. The New Project dialog appears.
2. On the New Project dialog, select **Web > Dynamic Web Project** and click **Next**. The New Dynamic Web Project wizard appears.
3. On the Dynamic Web Project page, enter the following information and click **Next**:
   - In the **Project name** field enter `Dali_Tutorial_Application`.
   - In the **Target Runtime** field, select your Web server configuration (for this tutorial, select **Apache Tomcat v5.5**), or click **New** to create a new configuration.
   - In the Configurations field, select **Custom**.

Figure 16 Dynamic Web Project Page of the New Dynamic Web Project Wizard



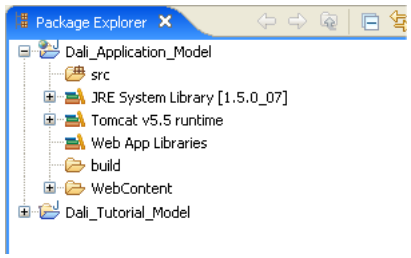
4. On the Project Facets page, verify that the Java facet uses version **5.0** and click **Finish**.

Figure 17 Project Facets Page



Eclipse creates an new Web project.

Figure 18 New Web Project



2.7.1 Add resources

Add the following resources to the **Dali_Tutorial_Application** project's `WebContent/WEB-INF/lib` directory:

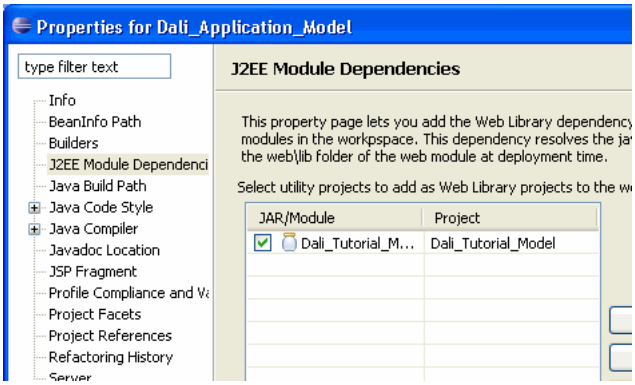
- JPA JARs (for this tutorial, use `toplink-essentials.jar` from the TopLink Essentials JPA)
- JSF JARs (for this tutorial, use the `jsf-api.jar` and `jsf-impl.jar` files from the JSF 1.1 library)
- Database JDBC JAR (for this tutorial, use `derbyclient.jar`)

2.7.2 Associate the model project

Use this procedure to associate the **Dali_Tutorial_Model** project with the **Dali_Tutorial_Application** project.

1. Right-click the **Dali_Tutorial_Application** project in the Explorer and select **Properties**.
2. Click **J2EE Module Dependencies**, select the **Dali_Tutorial_Model** project, and click **OK**.

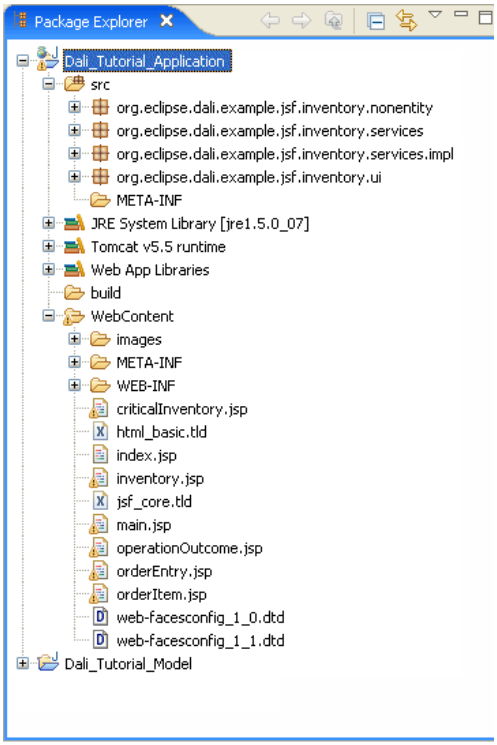
Figure 19 Project Build Path



2.7.3 Add the project files

The **Dali_Tutorial_Web.zip** (included with `http://www.eclipse.org/webtools/dali/docs/tutorial/jsf/Dali_Tutorial_Application.zip`) file contains the non-entity source files and Web content files for the dynamic Web project. Unzip the **Dali_Tutorial_Web.zip** file into your **Dali_Tutorial_Application** project's directory.

Figure 20 Dali_Tutorial_Application.zip Project



2.8 Deploy the application

You are now ready to deploy the tutorial application to your server.

1. Right-click the **Dali_Tutorial_Application** project in the Explorer and select **Run As > Run on Server**. The Run on Server dialog appears.

   If you have not yet defined a server, the Define a New Server dialog of the Run On Server wizard appears. Use this wizard to define a new server. For this tutorial, create a Tomcat v5.5 server.

2. Select the **Choose an existing server** option, select your sever, and click **Finish**.
   - The Server view shows the status of your sever and the tutorial application.
   - The Console view shows the status of the compilation, build, and deployment.

3. A new view opens, displaying the web page for the tutorial application.

Figure 21 Tutorial Application