Dali Java Persistence Tools User Guide

Copyright © 2006, 2009 Oracle. All rights reserved.

The Eclipse Foundation makes available all content in this plug-in ("Content"). Unless otherwise indicated below, the Content is provided to you under the terms and conditions of the Eclipse Public License Version 1.0 ("EPL"). A copy of the EPL is available at http://www.eclipse.org/legal/epl-v10.html. For purposes of the EPL, "Program" will mean the Content.

If you did not receive this Content directly from the Eclipse Foundation, the Content is being redistributed by another party ("Redistributor") and different terms and conditions may apply to your use of any object code in the Content. Check the Redistributor's license that was provided with the Content. If no such license exists, contact the Redistributor. Unless otherwise indicated below, the terms and conditions of the EPL still apply to any source code in the Content.
## Contents

1 Getting started
  1.1 Requirements and installation ................................................................. 1-1
  1.2 Dali quick start ......................................................................................... 1-2
  1.2.1 Creating a new JPA project ................................................................. 1-2
  1.2.2 Creating a Java persistent entity with persistent fields ......................... 1-3

2 Concepts
  2.1 Understanding Java persistence ............................................................... 2-1
  2.2 Understanding OR mappings ................................................................. 2-1
  2.3 Understanding EJB 3.0 Java Persistence API .......................................... 2-2
    2.3.1 The persistence.xml file ................................................................. 2-2
    2.3.2 The orm.xml file .............................................................................. 2-2

3 Tasks
  3.1 Creating a new JPA project ........................................................................ 3-1
  3.2 Converting a Java Project to a JPA Project ................................................ 3-5
  3.3 Creating a JPA Entity ................................................................................ 3-7
  3.4 Managing the persistence.xml file ........................................................... 3-10
    3.4.1 Synchronizing classes ...................................................................... 3-12
  3.5 Managing the orm.xml file ......................................................................... 3-12
    3.5.1 Creating an orm.xml file ................................................................. 3-13
    3.5.2 Working with orm.xml file ............................................................. 3-14
  3.6 Adding persistence to a class ..................................................................... 3-15
    3.6.1 Entity ............................................................................................... 3-15
    3.6.2 Embeddable ..................................................................................... 3-16
    3.6.3 Mapped superclass .......................................................................... 3-17
  3.7 Specifying additional tables ...................................................................... 3-18
  3.8 Specifying entity inheritance ................................................................. 3-19
  3.9 Creating Named Queries ......................................................................... 3-21
  3.10 Mapping an entity ................................................................................... 3-22
    3.10.1 Basic mapping ............................................................................... 3-22
    3.10.2 Embedded mapping ...................................................................... 3-23
    3.10.3 Embedded ID mapping .................................................................. 3-24
    3.10.4 ID mapping ................................................................................... 3-24
    3.10.5 Many-to-many mapping ............................................................... 3-26
4 Reference

4.1 Wizards................................................................. 4-1
4.1.1 Create New JPA Project wizard.......................... 4-1
4.1.1.1 New JPA Project page........................................ 4-1
4.1.1.2 Java Page......................................................... 4-2
4.1.1.3 JPA Facet page................................................ 4-2
4.1.2 Create JPA Entity wizard................................. 4-3
4.1.2.1 Entity Class page............................................. 4-4
4.1.2.2 Entity Properties page.................................... 4-4
4.1.3 Mapping File Wizard ....................................... 4-5
4.1.3.1 Mapping File................................................ 4-5
4.1.4 Generate Tables from Entities Wizard............... 4-6
4.1.5 Generate Entities from Tables Wizard............... 4-6
4.1.5.1 Select Tables................................................ 4-6
4.1.5.2 Table Associations........................................ 4-7
4.1.5.3 Customize Default Entity Generation............... 4-7
4.1.5.4 Customize Individual Entities.......................... 4-8
4.1.6 Create New Association................................. 4-8
4.1.6.1 Association Tables........................................ 4-8
4.1.6.2 Join Columns................................................. 4-8
4.1.6.3 Association Cardinality................................. 4-9
4.2 Property pages................................................ 4-9
4.2.1 JPA Details view (for entities).......................... 4-9
4.2.1.1 General information..................................... 4-9
4.2.1.2 Attribute overrides...................................... 4-10
4.2.1.3 Secondary table information......................... 4-10
4.2.1.4 Inheritance information.................. 4-10
4.2.1.5 Queries......................................................... 4-11
4.2.2 JPA Details view (for attributes)...................... 4-11
4.2.2.1 General information..................................... 4-12
4.2.2.2 Join Table Information................................. 4-14
4.2.2.3 Join Columns Information............................ 4-14
4.2.2.4 Primary Key Generation information............... 4-15
4.2.3 JPA Details view (for orm.xml)........................ 4-16
4.2.3.1 General information..................................... 4-16
This section provides information on getting started with the Java Persistence Tools.

- **Requirements and installation**
- **Dali quick start**

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

### 1.1 Requirements and installation

Before installing Dali, ensure that your environment meets the following minimum requirements:

- Eclipse 3.5 (http://www.eclipse.org/downloads)
- Java Runtime Environment (JRE) 1.5 (http://java.com)
- Eclipse Web Tools Platform (WTP) 3.1 (http://www.eclipse.org/webtools)
- Java Persistence API (JPA) for Java EE 5. For example, the EclipseLink implementation for JPA can be obtained from: http://www.eclipse.org/eclipselink/

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

Dali is included as part of WTP 3.1. No additional installation or configuration is required.

**Accessibility Features**

Dali supports the standard accessibility features in Eclipse, including the following:

- Navigating the user interface using the keyboard.
- Specifying general accessibility preferences for the editor.

See Accessibility Features in Eclipse in the Workbench User Guide for details.

**Help Accessibility**

The documentation and help contains markup to facilitate access by the disabled community. See Help Accessibility in the Workbench User Guide for details.

When using the help, be aware of the following:
1.2 Dali quick start

This section includes information to help you quickly start using Dali to create relational mappings between Java persistent entities and database tables.

- **Creating a new JPA project**
- **Creating a Java persistent entity with persistent fields**

1.2.1 Creating a new JPA project

This quick start shows how to create a new JPA project.

1. **Select File > New > Project.** The Select a Wizard dialog appears.
   
   **Tip:** You can also select the JPA perspective and then select File > New > JPA Project.

2. Select **JPA Project** and then click **Next.** The New JPA Project page appears.

3. Enter a **Project name** (such as **QuickStart**).

4. If needed, select the **Target Runtime** (such as **Apache Tomcat**) and configuration, such as **Utility JPA Project with Java 5.0** and then click **Next.** The Java source page appears.

   **Note:** The Target Runtime is not required for Java SE development.

5. If you have existing Java source files, add them to your classpath and then click **Next.** The JPA Facet page appears.

6. On the JPA Facet dialog, select your vendor-specific JPA platform (or select **Generic**), database connection (or create a new connection), JPA implementation library (such as EclipseLink), define how Dali should manage persistent classes, and then click **Finish.**

   **Tip:** Select **Override the Default Schema for Connection** if you require a schema other than the one that Dali derives from the connection information, which may be incorrect in some cases. Using this option, you can select a development time schema for defaults and validation.

Eclipse adds the project to the workbench and opens the JPA perspective.
Now that you have created a project with persistence, you can continue with creating a Java persistent entity with persistent fields.

### 1.2.2 Creating a Java persistent entity with persistent fields

This quick start shows how to create a new persistent Java entity. We will create an entity to associate with a database table. You will also need to add the ADDRESS table to your database.

1. Select the JPA project in the Navigator or Project Explorer and then click **New > Other**. The Select a Wizard dialog appears.

2. Select **JPA > Entity** and then click **Next**. The **Entity Class** page appears.

3. Enter the package name (such as `quickstart.demo.model`), the class name (such as `Address`) and then click **Next**. The **Entity Properties** page appears, which enables you to define the persistence fields, which you will map to the columns of a database table.

4. Use the Entity Fields dialog (invoked by clicking **Add** to add persistence fields to the Address class:
   ```java
   private Long id;
   private String city;
   private String country;
   private String stateOrProvince;
   private String postalCode;
   private String street;
   ```

   #### Note: You will also need to add the following columns to the ADDRESS database table:

   ```
   NUMBER(10,0) ADDRESS_ID (primary key)
   VARCHAR2(80) PROVINCE
   VARCHAR2(80) COUNTRY
   VARCHAR2(20) P_CODE
   VARCHAR2(80) STREET
   VARCHAR2(80) CITY
   ```

5. Click **Finish**. With the Create JPA Entity completed, Eclipse displays the **Address** entity in the JPA Structure view.
Address.java includes the @Entity annotation, the persistence fields, as well as getter and setter methods for each of the fields.

Figure 1–2  Address Entity in Address.java

```
package quickstart.demo.model;

@Entity
public class Address implements Serializable {

  // persistence fields

  // getters and setters
}
```

Eclipse also displays the Address entity in the JPA Structure view:

Figure 1–3  Address Entity in the JPA Structure View

1. Select the Address class in the Project Explorer view.
2. In the JPA Details view, notice that Dali has automatically associated the ADDRESS database table with the entity because they are named identically.

Note: Depending on your database connection type, you may need to specify the Schema.

Figure 1–4  JPA Details View for Address Entity
Now we are ready to map each fields in the Address class to a column in the database table.

1. Select the id field in the JPA Details view.
2. Right click id and then select Map As > id.
3. In the JPA Details view, select ADDRESS_ID in the Name field:

   **Figure 1–5  JPA Details View for the addressId Field**

   Eclipse adds the following annotations to the Address entity:

   ```java
   @Id
   @Column(name="ADDRESS_ID")
   ```

4. Map each of the following fields (as Basic mappings) to the appropriate database column:

<table>
<thead>
<tr>
<th>Field</th>
<th>Map As</th>
<th>Database Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>city</td>
<td>Basic</td>
<td>CITY</td>
</tr>
<tr>
<td>country</td>
<td>Basic</td>
<td>COUNTRY</td>
</tr>
</tbody>
</table>
Dali automatically maps some fields to the correct database column (such as the city field to the City column) if the names are identical.
This section contains an overview of concepts you should be familiar with when using Dali to create mappings for Java persistent entities.

- Understanding Java persistence
- Understanding OR mappings
- Understanding EJB 3.0 Java Persistence API

In addition to these sections, you should review the following resources for additional information:

- Eclipse Dali project: http://www.eclipse.org/webtools/dali
- Eclipse Web Tools Platform project: http://www.eclipse.org/webtools
- EclipseLink project: http://www.eclipse.org/eclipselink

2.1 Understanding Java persistence

*Persistence* refers to the ability to store objects in a database and use those objects with transactional integrity. In a J2EE application, data is typically stored and persisted in the data tier, in a relational database.

*Entity beans* are enterprise beans that contain persistent data and that can be saved in various persistent data stores. The entity beans represent data from a database; each entity bean carries its own identity. Entity beans can be deployed using *application-managed persistence* or *container-managed persistence*.

2.2 Understanding OR mappings

The Dali OR (object-relational) Mapping Tool allows you to describe how your entity objects *map* to the data source (or other objects). This approach isolates persistence information from the object model–developers are free to design their ideal object model, and DBAs are free to design their ideal schema.

These mappings transform an object data member type to a corresponding relational database data source representation. These OR mappings can also transform object data members that reference other domain objects stored in other tables in the database and are related through foreign keys.

You can use these mappings to map simple data types including primitives (such as `int`), JDK classes (such as `String`), and large object (LOB) values. You can also use them to transform object data members that reference other domain objects by way of
association where data source representations require object identity maintenance (such as sequencing and back references) and possess various types of multiplicity and navigability. The appropriate mapping class is chosen primarily by the cardinality of the relationship.

2.3 Understanding EJB 3.0 Java Persistence API

The Java 2 Enterprise Edition (J2EE) Enterprise JavaBeans (EJB) are a component architecture that you use to develop and deploy object-oriented, distributed, enterprise-scale applications. An application written according to the Enterprise JavaBeans architecture is scalable, transactional, and secure.

The EJB 3.0 Java Persistence API (JPA) improves the EJB architecture by reducing its complexity through the use of metadata (annotations) and specifying programmatic defaults of that metadata.

2.3.1 The persistence.xml file

The JPA specification requires the use of a persistence.xml file for deployment. This file defines the database and entity manager options, and may contain more than one persistence unit. To enable you to easily edit this information, Dali provides the persistence.xml Editor. Alternatively, you can use the Eclipse XML Editor to create and maintain this information. See "Managing the persistence.xml file" on page 3-10 for more information.

2.3.2 The orm.xml file

Although the JPA specification emphasizes the use of annotations to specify persistence, you can also use the orm.xml file to store this metadata. Dali enables you to create a stub orm.xml file for a JPA project using the Mapping File Wizard. See "Managing the orm.xml file" on page 3-12 for more information.

Note: The metadata must match the XSD specification of your selected JPA implementation.

Dali provides comprehensive support for configuring XML mapping files through the JPA Details view (for orm.xml) that is nearly identical to the annotation-based configuration in the Java source. Alternatively, you can also use the Eclipse XML Editor to create and maintain the metadata information in orm.xml.
This section includes detailed step-by-step procedures for accessing the Dali OR mapping tool functionality.

- Creating a new JPA project
- Converting a Java Project to a JPA Project
- Creating a JPA Entity
- Managing the persistence.xml file
- Managing the orm.xml file
- Adding persistence to a class
- Specifying additional tables
- Specifying entity inheritance
- Mapping an entity
- Generating entities from tables
- Validating mappings and reporting problems
- Modifying persistent project properties

3.1 Creating a new JPA project

Use this procedure to create a new JPA project.

1. From the Navigator or Project Explorer, select File > New > Project. The Select a wizard dialog appears.

   **Tip:** You can also select the JPA perspective and then select File > New > JPA Project.
2. Select JPA Project and then click Next. The New JPA Project page appears.

Figure 3–2 The JPA Project Page
3. Complete the fields on the New JPA Project page to specify the project name and location, target runtime, and pre-defined configuration.

**Note:** The Target Runtime is not required for Java SE development.

4. Click Next. The Java source page appears.

*Figure 3–3 The Java Source Page*

5. Click Add Folder to add existing Java source files to the project.

6. Click Next. JPA Facet page appears.
7. Complete the fields on the JPA Facet page to specify your vendor-specific platform, JPA implementation library, and database connection.

Click Manage libraries to create or update your JPA user libraries. Click Download libraries to obtain additional JPA implementation libraries.

If Dali derives the incorrect schema, select Override the Default Schema for Connection. Using this option, you can select a development time schema for defaults and validation.

If you clear the Create orm.xml option (which is selected by default), you can later add a mapping file to the project using the Mapping File Wizard.
Converting a Java Project to a JPA Project

3.2 Converting a Java Project to a JPA Project

Use this procedure to convert an existing Java project to a JPA project.

1. From the Navigator or Project explorer, right-click the Java project and then select Configure > Convert to JPA Project. The Project Facets page of the Modify Faceted Project wizard appears.

2. Change the Configuration to Utility JPA project.

3. Click Next. JPA Facet page appears.

8. Click Finish. You should now open the JPA Development perspective.
Converting a Java Project to a JPA Project

4. Complete the fields on the **JPA Facet page** to specify your vendor-specific platform, JPA implementation library, and database connection.

   Click **Manage libraries** to create or update your JPA user libraries. Click **Download libraries** to obtain additional JPA implementation libraries.

   If Dali derives the incorrect schema, select **Override the Default Schema for Connection**. Using this option, you can select a development time schema for defaults and validation.

   If you clear the **Create orm.xml** option (which is selected by default), you can later add a mapping file to the project using the **Mapping File Wizard**.

5. Click **Finish**.

   The Dali OR Mapping Tool adds the JPA implementation libraries to your project and creates the necessary **orm.xml** and **peristence.xml** files.
3.3 Creating a JPA Entity

Use this procedure to create a JPA entity:

1. From the Navigator or Project Explorer, select the JPA project and then File > New > Other. The Select a Wizard dialog appears.

![Selecting the Create a JPA Entity Wizard](image)

2. Select JPA > Entity and then click Next. The Entity Class page appears.
Complete this page as follows:

- Select the JPA project in the Project field.
- In the Source Folder field, select, or enter, the location of the JPA project’s src folder.
- Select, or enter, the name of the class package for this entity in the Java Package field.
- Enter the name of the Java class in the Class name field.
- If needed, enter, or select a superclass.
- If needed, complete the Inheritance section as follows (these properties are optional):
  - Accept the Entity option (the default) to create a Java class with the @Entity option.
  - Alternatively, select Mapped superclass (if you defined a super class).
  - Select Inheritance and then select one of the JSR 220 inheritance mapping strategies (SINGLE_TABLE, TABLE_PER_CLASS, JOINED).
  - Select Add to entity mappings in XML to create XML mappings in orm.xml, rather than annotations.

3. Click Next to proceed to the Entity Properties page where you define the persistent fields for the entity.
Figure 3–9  The Entity Properties Page

Alternatively, click Finish to complete the entity.

4. Complete the page as follows:
   1. If needed, enter a new name for the entity. Doing so results in adding a name attribute to the @Entity notation (@Entity(name="EntityName")).
   2. Accept Use default (the default setting) to use the default value for the name of the mapped table. Entering a different name results in adding the @Table notation with its name attribute defined as the new table (@Table(name="TableName")).

   **Note:** The Entity Name-related options are not available if you selected Mapped superclass on the Entity Class page

   3. Add persistence fields to the entity by clicking Add. The Entity Fields dialog appears.

   Figure 3–10  The Entity Fields Dialog

   4. Select a persistence type from the Type list. You can retrieve additional types using the Browse function.
   5. Enter the field name and then click OK. Repeat this procedure for each field.
3.4 Managing the persistence.xml file

When you create a project, Eclipse creates the META-INF\persistence.xml file in the project’s directory.

You can create a stub persistence.xml file in the META-INF directory when you create a JPA project (see "Creating a new JPA project"). You can manage this file either through the XML editor (see "Using the XML Editor to edit the persistence.xml file") or through the persistence.xml Editor.

### Note:
Depending on your JPA implementation (for example, EclipseLink), the following additional pages may be available in the persistence.xml Editor:

- **Customization**
  Use this page to define change-tracking and session customizer-related properties.
- **Caching**
  Use this page to define caching properties.
- **Logging**
  Use this page to define logging properties.
- **Options**
  Use this page to define session and target database properties.
- **Schema Generation**
  Use this page to define DDL-related properties.

For projects using the EclipseLink JPA implementation, the Connections page also includes JDBC connection pool properties.

If the project uses the Generic platform, then only the General, Connection, Properties and Source pages are available.

To use the persistence.xml Editor:

1. Open the persistence.xml file. The General page of the editor appears.

2. Use the General page to define the persistence.xml files
   <persistent-unit>-related attributes as well as the <provider>, and
   <class> elements (described in the following table).

   **Tip:** The persistence.xml Editor’s Source page enables you to view and edit the raw XML file.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Element Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter the name of the persistence unit.</td>
<td><code>&lt;persistence-unit name = &quot;&lt;Name&gt;&quot;</code></td>
</tr>
</tbody>
</table>
3. Use the Connection page to define the `<jta-data-source>` and `<non-jta-data-source>` elements as follows:

To configure the JTA (Java Transaction API) source used by the persistence provider:

1. Select JTA from the Transaction Type list.
2. Enter the global JNDI name of the data source.

To configure a non-JTA data source:

1. Select Resource Local from the Transaction Type list.
2. Enter the global JNDI name of the data source.

**Note:** Select Default() to use the data source provided by the container.

For projects using the Generic platform, you can also define the EclipseLink connection pool driver, connection pool driver, URL, user name and password.

4. Use the table in the Properties page to set the vendor-specific `<properties>` element.

To add `<property>` elements:

1. Click Add.
2. Enter the `<name>` and `<value>` attributes for the `<property>` element using the table’s Name and Value fields.

### Table: Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Element Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistence Provider</td>
<td>Enter the name of the persistence provider.</td>
<td><code>&lt;provider&gt;</code></td>
</tr>
<tr>
<td>Description</td>
<td>Enter a description for this persistence unit. This is an optional property.</td>
<td><code>&lt;description&gt;</code></td>
</tr>
<tr>
<td>Managed Classes</td>
<td>Add or remove the classes managed through the persistence unit.</td>
<td><code>&lt;class&gt;</code></td>
</tr>
<tr>
<td>Exclude Unlisted Classes</td>
<td>Select to include all annotated entity classes in the root of the persistence unit.</td>
<td><code>&lt;exclude-unlisted-classes&gt;</code></td>
</tr>
<tr>
<td>XML Mapping Files</td>
<td>Add or remove the object/relational mapping XML files define the classes managed through the persistence unit.</td>
<td><code>&lt;mapping-file&gt;</code></td>
</tr>
<tr>
<td>Exclude Unlisted Eclipse Link XML Mapping File</td>
<td>Select to include all annotated EclipseLink mapping files.</td>
<td><code>&lt;property name=&quot;eclipselink.exclude-eclipselink-orm&quot; value=&quot;true&quot;/&gt;</code></td>
</tr>
</tbody>
</table>
To remove a <property> element, select a defined property in the table and then click **Remove**.

---

**Note:** If the project uses the EclipseLink platform, the connection page also includes parameters for JDBC connection pooling.

### 3.4.1 Synchronizing classes

As you work with the classes in your Java project, you will need to update the persistence.xml file to reflect the changes.

Use this procedure to synchronize the persistence.xml file:

1. Right-click the persistence.xml file in the Project Explorer and select **Synchronize Classes**.

    **Note:** Use this function if you selected Annotated classes must be listed in the persistence.xml option in the JPA Facet page. In general, you do not have to use this function within the container.

**Figure 3–11  Synchronizing the persistence.xml File**

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

2. Use the Persistence XML Editor to continue editing the persistence.xml file.

### 3.5 Managing the orm.xml file

When creating a JPA project, (see "Creating a new JPA project") you can also create the orm.xml file that defines the mapping metadata and defaults.

Eclipse creates the META-INF\orm.xml file in your project’s directory:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="<PERSISTENCE_VERSION>"
    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="<PERSISTENCE_UNIT_NAME>">
    <provider="<PERSISTENCE_PROVIDER>"/>
  </persistence-unit>
</persistence>
```

---

Dali Java Persistence Tools User Guide
3.5.1 Creating an orm.xml file

If you opt not to create an `orm.xml` file when you create a JPA project, you can create one using the Mapping File Wizard.

Use this procedure to create an `orm.xml` file:

1. From the Navigator or Project Explorer, select **File > New > Other**. The Select a Wizard dialog appears.

   ![Figure 3–12 The Select a Wizard Dialog](image)

2. Select **Mapping File** and then click **Next**. The Mapping File page appears.

   If you are using EclipseLink, you can select **EclipseLink > EclipseLink Mapping File**.
3. Define the properties in the page and click Finish. The orm.xml file appears in the src directory of the selected JPA project. You can manage the orm.xml file using the JPA Details view or through the XML Editor. See also JPA Details view (for orm.xml).

### 3.5.2 Working with orm.xml file

You can work with the orm.xml by using the JPA Details view.

Use this procedure to work with the orm.xml file:

1. Right-click the orm.xml file in the Project Explorer and select Open.
2. In the JPA Structure view, select EntityMappings.
3. Use the JPA Details view to configure the entity mapping and persistence unit defaults.
3.6 Adding persistence to a class

You can make a Java class into one of the following persistent types:

- **Entity**
- **Embeddable**
- **Mapped superclass**

### 3.6.1 Entity

An **Entity** is a persistent domain object.

An entity can be:
- Abstract or concrete classes. Entities may also extend non-entity classes as well as entity classes, and non-entity classes may extend entity classes.

An entity must have:
- A no-arg constructor (public or protected); the entity class may have other constructors as well.

Each persistent entity must be mapped to a database table and contain a primary key. Persistent entities are identified by the `@Entity` annotation.

Use this procedure to add persistence to an existing entity:

1. Open the Java class in the Project Explorer.
2. Select the class in the JPA Structure view.
3. In the JPA Details view, click the mapping type hyperlink to access the Mapping Type Selection dialog. In the following figure, clicking *entity* invokes the dialog from the JPA Details View.

*Figure 3–15  The Mapping Type Hyperlink*
4. Select Entity from the Mapping Type Selection dialog and then click OK.

**Figure 3–16 The Mapping Type Selection Dialog**

5. Complete the remaining JPA Details view (for entities).

3.6.2 Embeddable

An **Embeddable** class is a class whose instances are stored as part of an owning entity; it shares the identity of the owning entity. Each field of the embedded class is mapped to the database table associated with the owning entity.

To override the mapping information for a specific subclass, use the `@AttributeOverride` annotation for that specific class.

An embeddable entity is identified by the `@Embeddable` annotation.

Use this procedure to add embeddable persistence to an existing entity:

1. Open the Java class in the Project Explorer.
2. Select the class in the JPA Structure view.
3. Click the mapping type hyperlink to open the Mapping Type Selection dialog.
4. Select Embeddable and then click OK.
5. Complete the remaining JPA Details view (for entities).

### 3.6.3 Mapped superclass

An entity that extends a **Mapped Superclass** class inherits the persistent state and mapping information from a superclass. You should use a mapped superclass to define mapping information that is common to multiple entity classes.

A mapped superclass can be:
- Abstract or concrete classes

A mapped superclass cannot be:
- Be queried or passed as an argument to Entity-Manager or Query operations
- Be the target of a persistent relationship

A mapped superclass does not have a defined database table. Instead, its mapping information is derived from its superclass. To override the mapping information for a specific subclass, use the `@AttributeOverride` annotation for that specific class.

A mapped superclass is identified by the `@MappedSuperclass` annotation.

Use this procedure to add Mapped Superclass persistence to an existing entity:

1. Open the Java class in the Project Explorer.
2. Select the class in the JPA Structure view.
3. In the JPA Details view, click the mapping type hyperlink to open the Mapping Type Selection dialog.
4. Select **Mapped Superclass** and then **OK**.
5. Complete the remaining JPA Details view (for entities).

### 3.7 Specifying additional tables

Add a secondary table annotation to an entity if its data is split across more than one table.

To add a secondary table to the entity,

1. Select the entity in the Project Explorer.
2. In the JPA Details view, select the Secondary Tables information.

3. Click Add to associate an additional table with the entity. The Edit Secondary Table dialog appears.
4. Select the Name, Catalog, and Schema of the additional table to associate with the entity.

   Eclipse adds the following annotations the entity:

   ```java
   @SecondaryTable(name='NAME', catalog = 'CATALOG', schema = 'SCHEMA')
   ```

   To override the default primary key:
1. Enable the **Overwrite default** option, then click **Add** to specify a new primary key join column. The Create New Primary Key Join Column appears.

2. Select the **Name**, **Referenced column name**, **Table**, and **Column definition** of the primary key for the entity.

   Eclipse adds the following annotations the entity:

   ```java
   @SecondaryTable(name="NAME", catalog = "CATALOG", schema = "SCHEMA",
   pkJoinColumns = {@PrimaryKeyJoinColumn(name="id", referencedColumnName = "id"),
   @PrimaryKeyJoinColumn(name="NAME", referencedColumnName = "REFERENCED COLUMN NAME",
   columnDefinition = "COLUMN DEFINITION")})
   ```

3.8 **Specifying entity inheritance**

An entity may inherit properties from other entities. You can specify a specific strategy to use for inheritance.

Use this procedure to specify inheritance (@Inheritance) for an existing entity (@Entity):

1. Select the entity in the Project Explorer.

2. In the JPA Details view, select the **Inheritance** information.

![Figure 3–20 Specifying Inheritance](image)

3. In the **Strategy** list, select one of the following the inheritance strategies:
   - A single table (default)
   - Joined table
   - One table per class

4. Use the following table to complete the remaining fields on the tab. See "Inheritance information" on page 4-10 for additional details.
Specifying entity inheritance

Eclipse adds the following annotations the entity field:

```java
@Inheritance(strategy=InheritanceType.<INHERITANCE_STRATEGY>)
@DiscriminatorColumn(name="<DISCRIMINATOR_COLUMN>",
                     discriminatorType=<DISCRIMINATOR_TYPE>)
@DiscriminatorValue(value="<DISCRIMINATOR_VALUE>")
@PrimaryKeyJoinColumn(name="<JOIN_COLUMN_NAME>",
                       referencedColumnName="<REFERENCED_COLUMN_NAME>")
```

The following figures illustrates the different inheritance strategies.

**Figure 3–21 Single Table Inheritance**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discriminator Column</td>
<td>Name of the discriminator column when using a Single or Joined inheritance strategy. This field corresponds to the @DiscriminatorColumn annotation.</td>
<td></td>
</tr>
<tr>
<td>Discriminator Type</td>
<td>Set the discriminator type to Char or Integer (instead of its default: String). The Discriminator Value must conform to this type.</td>
<td>String</td>
</tr>
<tr>
<td>Discriminator Value</td>
<td>Specify the discriminator value used to differentiate an entity in this inheritance hierarchy. The value must conform to the specified Discriminator Type. This field corresponds to the @DiscriminatorValue annotation.</td>
<td></td>
</tr>
<tr>
<td>Override Default</td>
<td>Use this field to specify custom primary key join columns. This field corresponds to the @PrimaryKeyJoinColumn annotation.</td>
<td></td>
</tr>
</tbody>
</table>
3.9 Creating Named Queries

Named queries improve application performance because they are prepared once and they (and all of their associated supporting objects) can be efficiently reused thereafter, making them well suited for complex and frequently executed operations. Named queries use the JPA query language for portable execution on any underlying database; named native queries use the SQL language native to the underlying database.

Use this procedure to add `@NamedQuery` and `@NamedNativeQuery` annotations to the entity.

To create a named query:

1. Select the entity in the Project Explorer.
2. In the JPA Details view, expand Queries.
3. Click Add for a named query, or Add Native for a native query.
4. In the dialog that appears, enter the name of the query in the Name field and then click OK.
5. Enter the query in the Query field.
6. To add a Query hint, click Add.
3.10 Mapping an entity

Dali supports the following mapping types for Java persistent entities:

- Basic mapping
- Embedded mapping
- Embedded ID mapping
- ID mapping
- Many-to-many mapping
- Many-to-one mapping
- One-to-many mapping
- One-to-one mapping
- Transient mapping
- Version mapping

Additional mapping types (such as Basic Collection mappings) may be available when using Dali with EclipseLink.

3.10.1 Basic mapping

Use a Basic Mapping to map an attribute directly to a database column. Basic mappings may be used only with the following attribute types:

- Java primitive types and 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[]
- Character[]
- enums
- any other type that implements Serializable

To create a basic mapping:
1. In the JPA Structure view, right-click the field to map. Select Map As > Basic. The JPA Details view (for attributes) displays the properties for the selected field.
2. Use this table to complete the remaining fields on the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Map Hyperlink</td>
<td>Defines this mapping as a Basic Mapping. This corresponds to the @Basic annotation.</td>
<td>Basic</td>
</tr>
<tr>
<td>Column</td>
<td>The database column mapped to the entity attribute. See &quot;Column&quot; on page 4-12 for details.</td>
<td>By default, the Column is assumed to be named identically to the attribute and always included in the INSERT and UPDATE statements.</td>
</tr>
<tr>
<td>Table</td>
<td>Name of the database table.</td>
<td></td>
</tr>
<tr>
<td>Fetch</td>
<td>Defines how data is loaded from the database. See &quot;Fetch Type&quot; on page 4-13 for details.</td>
<td>Eager</td>
</tr>
<tr>
<td>Optional</td>
<td>Specifies if this field is can be null.</td>
<td>Yes</td>
</tr>
<tr>
<td>Temporal</td>
<td>Specifies the type of data. See &quot;Temporal&quot; on page 4-13 for details.</td>
<td></td>
</tr>
<tr>
<td>Lob</td>
<td>Specifies if this is a large objects (BLOB or CLOB). See &quot;Lob&quot; on page 4-13 for details.</td>
<td></td>
</tr>
</tbody>
</table>

Eclipse adds the following annotations to the field:

```java
@Column(name="<COLUMN_NAME>", table="<COLUMN_TABLE>",
    insertable=<INSERTABLE>, updatable=<UPDATABLE>)
@Basic(fetch=<FETCH_TYPE>, optional = <OPTIONAL>)
@Temporal(<TEMPORAL>)
```

### 3.10.2 Embedded mapping

Use an Embedded Mapping to specify a persistent field or property of an entity whose value is an instance of an embeddable class.
1. In the JPA Structure view, right-click the field to map.
2. Select **Map as > Embedded**. The JPA Details view (for attributes) displays the properties for the selected field.

3. Use this table to complete the remaining fields on the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Mapping Hyperlink</td>
<td>Defines this mapping as a <strong>Embedded</strong>. This corresponds to the @Embedded annotation.</td>
<td>Embedded</td>
</tr>
<tr>
<td>Attribute Overrides</td>
<td>Specify to override the default mapping of an entity’s attribute. Select <strong>Override Default</strong>.</td>
<td></td>
</tr>
<tr>
<td>Columns</td>
<td>The database column (and its table) mapped to the entity attribute. See &quot;Column“ on page 4-12 for details.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Name – Name of the database column.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Table – Name of the database table.</td>
<td></td>
</tr>
</tbody>
</table>

Eclipse adds the following annotations to the field:

```java
@Embedded
@AttributeOverride(column=@Column(table="<COLUMN_TABLE>", name = "<COLUMN_NAME>"))
```

### 3.10.3 Embedded ID mapping

Use an **Embedded ID Mapping** to specify the primary key of an embedded ID. These mappings may be used with a **Embeddable** entities.

1. In the JPA Structure view, select the field to map.

2. Right-click the field and then select **Map As > Embedded Id**. The JPA Details view (for attributes) displays the properties for the selected field.

3. Use this table to complete the remaining fields on the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Mapping Hyperlink</td>
<td>Defines this mapping as a <strong>Embedded Id</strong>. This corresponds to the @EmbeddedId annotation.</td>
<td>Embedded Id</td>
</tr>
</tbody>
</table>

Eclipse adds the following annotations to the field:

```java
@EmbeddedId
```

### 3.10.4 ID mapping

Use an **ID Mapping** to specify the primary key of an entity. ID mappings may be used with a **Entity** or **Mapped superclass**. Each **Entity** must have an ID mapping.

1. In the JPA Structure view, select the field to map.
2. Right click the field and then select **Map as > ID**. The JPA Details view (for attributes) displays the properties for the selected.

3. Use this table to complete the **General information** fields in the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Mapping Hyperlink</td>
<td>Defines this mapping as an <strong>ID Mapping</strong>. This field corresponds to the <code>@Id</code> annotation.</td>
<td>ID</td>
</tr>
<tr>
<td>Column</td>
<td>The database column mapped to the entity attribute. See &quot;Column&quot; on page 4-12 for details.</td>
<td>By default, the Column is assumed to be named identically to the attribute.</td>
</tr>
<tr>
<td>Table</td>
<td>The database table mapped to the entity attribute.</td>
<td>By default, the Table is assumed to be identical to the table associated with the entity.</td>
</tr>
<tr>
<td>Temporal</td>
<td>Specifies the type of data. See &quot;Temporal&quot; on page 4-13 for details.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Date</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Timestamp</td>
<td></td>
</tr>
</tbody>
</table>

4. Use this table to complete the fields in **Primary Key Generation information** area in the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Key Generation</td>
<td>These fields define how the primary key is generated.</td>
<td></td>
</tr>
<tr>
<td>Strategy</td>
<td>See &quot;Primary Key Generation&quot; on page 4-15 for details.</td>
<td>Auto</td>
</tr>
<tr>
<td></td>
<td>- Auto</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Sequence</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Identity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Table</td>
<td></td>
</tr>
<tr>
<td>Generator Name</td>
<td>Name of the primary key generator specified in the <strong>Strategy</strong></td>
<td></td>
</tr>
</tbody>
</table>

Additional fields will appear in the **Primary Key Generation information** area, depending on the selected Strategy. See "JPA Details view (for attributes)" on page 4-11 for additional information.

Eclipse adds the following annotations to the field:

```java
@Id
@Column(name="<COLUMN_NAME>", table="<TABLE_NAME>", insertable=<INSERTABLE>, updatable=<UPDATABLE>)
@Temporal(<TEMPORAL>)
@GeneratedValue(strategy=GeneratorType.<STRATEGY>, generator="<GENERATOR_NAME>")
@TableGenerator(name="<TABLE_GENERATOR_NAME>", table="<TABLE_GENERATOR_TABLE>", pkColumnName = "<TABLE_GENERATOR_PK>", valueColumnName = "<TABLE_GENERATOR_VALUE_COLUMN>", pkColumnValue = "<TABLE_GENERATOR_PK_COLUMN_VALUE>")
@SequenceGenerator(name="<SEQUENCE_GENERATOR_NAME>",
```
3.10.5 Many-to-many mapping

Use a Many-to-Many Mapping to define a many-valued association with many-to-many multiplicity. A many-to-many mapping has two sides: the owning side and non-owning side. You must specify the join table on the owning side. For bidirectional mappings, either side may be the owning side.

1. In the JPA Structure view, select the field to map.
2. Right-click the field and then select Map As > Many-to-Many. The JPA Details view (for attributes) displays the properties for the selected field.
3. Use this table to complete the General information fields of the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping Entity Hyperlink</td>
<td>Defines this mapping as a Many to Many Mapping.</td>
<td>Many to Many</td>
</tr>
<tr>
<td></td>
<td>This field corresponds to the @ManyToMany annotation.</td>
<td></td>
</tr>
<tr>
<td>Target Entity</td>
<td>The entity to which this attribute is mapped.</td>
<td>null</td>
</tr>
<tr>
<td></td>
<td>You do not need to explicitly specify the target entity, since it can be</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inferred from the type of object being referenced.</td>
<td></td>
</tr>
<tr>
<td>Fetch</td>
<td>Defines how data is loaded from the database. See “Fetch Type” on page 4-13 for details.</td>
<td>Lazy</td>
</tr>
<tr>
<td></td>
<td>■ Eager</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Lazy</td>
<td></td>
</tr>
<tr>
<td>Mapped By</td>
<td>The database field that owns the relationship.</td>
<td></td>
</tr>
<tr>
<td>Order By</td>
<td>Specify the default order for objects returned from a query. See “Order By” on page 4-14 for details.</td>
<td>No ordering</td>
</tr>
<tr>
<td></td>
<td>■ No ordering</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Primary key</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Custom</td>
<td></td>
</tr>
</tbody>
</table>

4. Use this table to complete the fields in the Join Table Information area in the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the join table that contains the foreign key column.</td>
<td>You must specify the join table on the owning side.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>By default, the name is assumed to be the primary tables associated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>with the entities concatenated with an underscore.</td>
</tr>
</tbody>
</table>
5. To add a new Join or Inverse Join Column, click **Add**.

   To edit an existing Join or Inverse Join Column, select the field to and click **Edit**.

Eclipse adds the following annotations to the field:

```java
@JoinTable(joinColumns=@JoinColumn(name="<JOIN_COLUMN>"),
           name = "<JOIN_TABLE_NAME>")
@ManyToMany(cascade=CascadeType.<CASCADE_TYPE>, fetch=FetchType.<FETCH_TYPE>,
           targetEntity=<TARGET_ENTITY>, mappedBy = "<MAPPED_BY>"
@OrderBy("<ORDER_BY>")
```

### 3.10.6 Many-to-one mapping

Use a **Many-to-One** mapping to defines a single-valued association to another entity class that has many-to-one multiplicity.

1. In the JPA Structure view, select the field to map.
2. Right click the field and then select **Map As > Many-to-One**. The JPA Details view (for attributes) displays the properties for the selected.
3. Use this table to complete the **General information** fields JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping Entity Hyperlink</td>
<td>Defines mapping as <strong>Many-to-One</strong>. This corresponds to the @ManyToOne annotation.</td>
<td>Many-to-One</td>
</tr>
<tr>
<td>Target Entity</td>
<td>The entity to which this attribute is mapped.</td>
<td>null</td>
</tr>
<tr>
<td>Fetch</td>
<td>Defines how data is loaded from the database. See &quot;Fetch Type&quot; on page 4-13 for details.</td>
<td>Eager</td>
</tr>
<tr>
<td>Cascade</td>
<td>See &quot;Cascade Type&quot; on page 4-13 for details.</td>
<td>Default</td>
</tr>
</tbody>
</table>
3.10.7 One-to-many mapping

Use a **One-to-Many Mapping** to define a relationship with one-to-many multiplicity.

1. In the **JPA Structure view**, select the field to map.
2. Right-click the field and then select **Map As > One-to-many**. The **JPA Details view (for attributes)** displays the properties for the selected.
3. Use this table to complete the **General information** fields JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping Entity</td>
<td>Defines mapping as <strong>One-to-Many</strong>. This corresponds to the @OneToMany annotation.</td>
<td>One-to-Many</td>
</tr>
<tr>
<td>Type Hyperlink</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target Entity</td>
<td>The entity to which this attribute is mapped.</td>
<td></td>
</tr>
<tr>
<td>Cascade</td>
<td>See &quot;Cascade Type&quot; on page 4-13 for details.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Default</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- All</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Persist</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Merge</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Remove</td>
<td></td>
</tr>
<tr>
<td>Fetch</td>
<td>Defines how data is loaded from the database. See &quot;Fetch Type&quot; on page 4-13 for details.</td>
<td>Eager</td>
</tr>
<tr>
<td></td>
<td>- Eager</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Lazy</td>
<td></td>
</tr>
</tbody>
</table>
4. Use this table to complete the Join Table Information fields in the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped By</td>
<td>The database field that owns the relationship.</td>
<td></td>
</tr>
<tr>
<td>Order By</td>
<td>Specify the default order for objects returned from a query. See &quot;Order By&quot; on page 4-14 for details.</td>
<td>No ordering</td>
</tr>
<tr>
<td></td>
<td>■ No ordering</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Primary key</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Custom</td>
<td></td>
</tr>
</tbody>
</table>

Eclipse adds the following annotations to the field:

```java
@OneToMany(targetEntity=<TARGET_ENTITY>)
@JoinColumn(name=<COLUMN>)
```

```java
@OneToMany(targetEntity=<TARGET_ENTITY>.class, cascade=CascadeType.<CASCADE_TYPE>, fetch = FetchType.<FETCH_TYPE>, mappedBy = "<MAPPED_BY>")
@OrderBy("<ORDER_BY>")
@JoinTable(name="<JOIN_TABLE_NAME>", joinColumns=@JoinColumn(name="<JOIN_COLUMN_NAME>", referencedColumnName="<JOIN_COLUMN_REFERENCED_COLUMN>"), inverseJoinColumns=@JoinColumn(name="<INVERSE_JOIN_COLUMN_NAME>", referencedColumnName="<INVERSE_JOIN_COLUMN_REFERENCED_COLUMN>"))
```

### 3.10.8 One-to-one mapping

Use a One-to-One Mapping to define a relationship with one-to-many multiplicity.

1. In the JPA Structure view, select the field to map.
2. Right-click the field and then select Map As > One-to-One. The JPA Details view (for attributes) displays the properties for the selected.
3. Use this table to complete the General information fields in the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped Entity Hyperlink</td>
<td>Defines mapping as One-to-One. This corresponds to the @OneToOne annotation.</td>
<td>One-to-One</td>
</tr>
</tbody>
</table>
4. Use this table to complete the Join Columns Information fields in the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| Target Entity | The entity to which this attribute is mapped.                              | null
You do not need to explicitly specify the target entity, since it can be inferred from the type of object being referenced. |
| Fetch Type  | Defines how data is loaded from the database. See "Fetch Type" on page 4-13 for details. |
|             | ■ Eager                                                                     | Eager                                                                  |
|             | ■ Lazy                                                                      |                                                                         |
| Mapped By   | The database field that owns the relationship.                              |                                                                         |

Eclipse adds the following annotations to the field:

```java
@OneToOne(targetEntity=<TARGET_ENTITY>, cascade=CascadeType.<CASCADE_TYPE>,
  fetch = FetchType.<FETCH_TYPE>, mappedBy = "<MAPPED_BY>")
@JoinColumn(name="<JOIN_COLUMN_NAME>", referencedColumnName="<JOIN_COLUMN_REFERENCED_COLUMN>", insertable = <INSERTABLE>,
  updatable = <UPDATABLE>)
```

3.10.9 Transient mapping

Use the Transient Mapping to specify a field of the entity class that is *not* persistent.

To create a transient mapping:

1. In the JPA Structure view, select the field to map.
2. Right-click the field and then select **Map As Transient**. The JPA Details view (for attributes) displays the properties for the selected.

Eclipse adds the following annotation to the field:

```java
@Transient
```

3.10.10 Version mapping

Use a **Version Mapping** to specify the field used for optimistic locking. If the entity is associated with multiple tables, you should use a version mapping only with the
primary table. You should have only a single version mapping per persistent entity. Version mappings may be used only with the following attribute types:

- `int`
- `Integer`
- `short`, `Short`
- `long`, `Long`
- `Timestamp`

To create a version mapping:

1. In the **JPA Structure view**, select the field to map.
2. Right-click the field and then select **Map As > Version**. The **JPA Details view** (for attributes) displays the properties for the selected.
3. Use this table to complete the remaining fields in the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped Entity Hyperlink</td>
<td>Defines the mapping as Version. This corresponds to the @Version annotation.</td>
<td>Version</td>
</tr>
<tr>
<td>Column</td>
<td>The database column mapped to the entity attribute. See &quot;Column&quot; on page 4-12 for details.</td>
<td>By default, the Column is assumed to be named identically to the attribute and always included in the <code>INSERT</code> and <code>UPDATE</code> statements.</td>
</tr>
<tr>
<td>Table</td>
<td>Name of the database table. This must be the primary table associated with the attribute's entity.</td>
<td></td>
</tr>
<tr>
<td>Temporal</td>
<td>Specifies the type of data. See &quot;Temporal&quot; on page 4-13 for details.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Date</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Timestamp</td>
<td></td>
</tr>
</tbody>
</table>

Eclipse adds the following annotations to the field:

```java
@Version
@Column(table='<COLUMN_TABLE>', name='<COLUMN_NAME>')
```

### 3.11 Generating entities from tables

Use this procedure to generate Java persistent entities from database tables. You must create a JPA project and establish a database connection before generating persistent entities. See "Creating a new JPA project" on page 3-1 for more information.

1. Right-click the JPA project in the Project Explorer and select **JPA Tools > Generate Entities from Tables**.
2. On the **Select Tables** page, select your database connection and schema. To create a new database connection, click **Add connection**. If you are not currently connected to the database, the Database Connection page appears. Select your database connection and schema, and click **Reconnect**.

**Figure 3–24 Generating Entities**

![Image of Generate Entities from Tables]

3. After selecting a schema, select the tables from which to generate Java persistent entities and click **Next**.

4. On the **Table Associations** page, select the associations to generate. You can specify to generate specific references for each association. To create a new association, click **Add Association**. Use the **Create New Association** wizard to define the association.

**Figure 3–25 Select Tables**

![Image of Select Tables]
5. After editing the table associations, click **Next**.

6. On the **Customize Default Entity Generation** page, customize the mapping and class information for each generated entity.
7. After customizing the mappings, click **Next**.

8. On the **Customize Individual Entities** page, review the mapping and class information for each entity that will be generated, then click **Finish**.
Eclipse creates a Java persistent entity for each database table. Each entity contains fields based on the table’s columns. Eclipse will also generate entity relationships (such as one-to-one) based on the table constraints. Figure 3–29 illustrates how Eclipse generates entities from tables.
3.12 Generating tables from entities

When using a vendor-specific platform, you can create a DDL script from your persistent entities.

---

**Note:** The DDL script with DROP existing tables on the database and CREATE new tables, based on the entities in your project.

---

To generate a DDL script:

Right-click the JPA project in the Project Explorer and select JPA Tools > Generate Tables from Entities.

3.13 Validating mappings and reporting problems

Errors and warnings on persistent entities and mappings are indicated with a red error or yellow warning next to the resource with the error, as well as the parent containers up to the project.

*Figure 3–30 Sample Errors and Warnings*

This section contains information on the following:

- Error messages
- Warning messages

3.13.1 Error messages

This section contains information on error messages (including how to resolve the issue) you may encounter while working with Dali.

**Attribute "<ATTRIBUTE__NAME>" has invalid mapping type in this context**

The mapped attribute is invalid. Either change the mapping type or change the entity type.

See "Mapping an entity" on page 3-22 for more information.
Attribute "<ATTRIBUTE_NAME>" cannot be resolved.
Dali cannot map the attribute to a database table and column. Verify that you database connection information is correct.
See "Creating a new JPA project" on page 3-1 for more information.

Class "<CLASS_NAME>" is not annotated as a persistent class.
The class has not been identified as a persistent class. Configure the class as an Entity, Mapped Superclass, or Embeddable persistent entity.
See "Adding persistence to a class" on page 3-15.

Column "<COLUMN_NAME>" cannot be resolved.
You mapped an entity’s field to an incorrect or invalid column in the database table. By default, Dali will attempt to map each field in the entity with an identically named row in the database table. If the field’s name differs from the row’s name, you must explicitly create the mapping.
Map the field to a valid row in the database table as shown in "Mapping an entity" on page 3-22.

Duplicate class "<CLASS_NAME>".
You created to persistence classes with the same name. Each Java class must have a unique name. See "Adding persistence to a class" on page 3-15 for more information.

Entity does not have an Id or Embedded Id.
You created a persistent entity without identifying its primary key. A persistent entity must have a primary key field designated with an @Id or @EmbeddedId annotation.
Add an ID mapping to the entity as shown in "ID mapping" on page 3-24 or "Embedded ID mapping" on page 3-24.

Multiple persistence.xml files in project.
You created a JPA project with more than one persistence.xml file. Each JPA project must contain a single persistence.xml file.
See "Managing the persistence.xml file" on page 3-10 for more information.

No persistence unit defined.
There is no persistence unit defined in the persistence.xml file. Use the <persistence-unit name="<PERSISTENCE_UNIT_NAME>" tag to define the persistent unit.
See "Managing the orm.xml file" on page 3-12 for more information.

No persistence.xml file in project.
You created a JPA project without a persistence.xml file. Each JPA project must contain a single persistence.xml file.
See "Managing the persistence.xml file" on page 3-10 for more information.

Referenced column "<COLUMN_NAME>" in join column "<COLUMN_NAME>" cannot be resolved.
The column that you selected to join a relationship mapping does not exist on the database table. Either select a different column on the Join Table Information or create the necessary column on the database table.
Modifying persistent project properties

See "JPA Details view (for attributes)" on page 4-11 for more information.

**Schema "<SCHEMA_NAME>" cannot be resolved for table/join table "<TABLE_NAME>".**
Define the default database schema information in the persistence unit.
See "Managing the orm.xml file" on page 3-12 for more information.

**Table "<TABLE_NAME>" cannot be resolved.**
You associated a persistent entity to an incorrect or invalid database table. By default, Dali will attempt to associate each persistent entity with an identically named database table. If the entity’s name differs from the table’s name, you must explicitly create the association.

Associate the entity with a valid database table as shown in "Adding persistence to a class" on page 3-15.

**Unresolved generator "<GENERATOR_NAME>" is defined in persistence unit.**
You created a persistence entity that uses sequencing or a table generator, but did not define the generator in the persistence unit. Either define the generator by using an annotation or including it in the XML mapping file.

### 3.13.2 Warning messages

This section contains information on warning messages (including how to resolve the issue) you may encounter while working with Dali.

**Connection "<CONNECTION_NAME>" is not active. No validation will be done against the data source.**
The database connection you specified to use with the JPA project is not active. The JPA project requires an active connection.

**No connection specified for the project. No data-specific validation will be performed.**
You created a JPA project without specifying a database connection. The JPA project requires an active connection.

See "Creating a new JPA project" on page 3-1 or "Modifying persistent project properties" on page 3-38 for information on specifying a database connection.

### 3.14 Modifying persistent project properties

Each persistent project must be associated with a database connection. To create a new database connection, click **Database Connection** use the New Connection wizard.

Use this procedure to modify the vendor-specific platform and database connection associated with your JPA project.
1. Right-click the project in the Explorer view and select **Properties**. The Properties page appears.

*Figure 3–31 The Properties Page*

2. Complete each field on the **Project Properties page – Java Persistence Options** click **OK**.

See "Project Properties page – Java Persistence Options" on page 4-30 for more information.
This section includes detailed help information for each of the following elements in the Dali OR Mapping Tool:

- Wizards
- Property pages
- Preferences
- Dialogs
- JPA Development perspective
- Icons and buttons
- Dali Developer Documentation

4.1 Wizards

This section includes information on the following wizards:

- Create New JPA Project wizard
- Create JPA Entity wizard
- Generate Tables from Entities Wizard
- Generate Entities from Tables Wizard
- Create New Association

4.1.1 Create New JPA Project wizard

The Create New JPA Project wizard allows you to create a new Java project using JPA. The wizard consists of the following pages:

- New JPA Project page
- Java Page
- JPA Facet page

4.1.1.1 New JPA Project page

This table lists the properties available on the New JPA Project page of the Create New JPA Project wizard.
### 4.1.1.2 Java Page

This table lists the properties available on the Java page of the Create New JPA Project wizard.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project name</td>
<td>Name of the Eclipse JPA project.</td>
<td></td>
</tr>
<tr>
<td>Project contents</td>
<td>Location of the workspace in which to save the project.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unselect The Use Default option and click Browse to select a new location.</td>
<td></td>
</tr>
<tr>
<td>Target runtime</td>
<td>Select a pre-defined target for the project.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Click New to create a new environment with the New Server Runtime wizard.</td>
<td></td>
</tr>
<tr>
<td>Configurations</td>
<td>Select a project configuration with pre-defined facets.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Select &lt;custom&gt; to manually select the facets for this project.</td>
<td></td>
</tr>
<tr>
<td>EAR membership</td>
<td>Specify if this project should be included in an EAR file for deployment.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Select the EAR Project Name, or click New to create a new EAR project.</td>
<td></td>
</tr>
<tr>
<td>Working sets</td>
<td>Specify if this project should be included in an existing working set.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The drop down field shows a list of previous selected working sets.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Select Add project to working sets, then select a working set in which to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>add this project.</td>
<td></td>
</tr>
<tr>
<td>Source folders on build</td>
<td>Click Add Folder to select an existing Java source folder to add to this</td>
<td>EclipseLink</td>
</tr>
<tr>
<td>path</td>
<td>project.</td>
<td></td>
</tr>
<tr>
<td>Default output folder</td>
<td>Specify the location of the .class files.</td>
<td>build\classes</td>
</tr>
</tbody>
</table>

### 4.1.1.3 JPA Facet page

This table lists the properties available on the JPA Facet page of the Create New JPA Project wizard.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>Vendor-specific JPA implementation.</td>
<td>EclipseLink</td>
</tr>
<tr>
<td>JPA Implementation</td>
<td>Select a specific JPA library configuration.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Click Manage libraries to create or update a user library.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Click Download libraries to download a specific library configuration.</td>
<td></td>
</tr>
</tbody>
</table>
4.1.2 Create JPA Entity wizard

The Create JPA wizard enables you to quickly add an entity and also add persistence fields to that entity. In addition, this wizard adds the accessor methods (getter and setter) in the class file. The wizard consists of the following pages:

- Entity Class page
- Entity Properties page

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Select <strong>User Library</strong> to select from the available user-defined or downloaded libraries. If you select Disable, you must manually include the JPA implementation library on the project classpath.</td>
<td>User Library</td>
</tr>
<tr>
<td>Include libraries with this application</td>
<td>Specify if the selected libraries are included when deploying the application.</td>
<td>Selected</td>
</tr>
<tr>
<td>Connection</td>
<td>Select the database connection to use with the project. Dali requires an active database connection to use and validate the persistent entities and mappings. Click <strong>Add connection</strong> to create a new database connection.</td>
<td></td>
</tr>
<tr>
<td>Override default schema from connection</td>
<td>Select a schema other than the default one that is derived from the connection information. Use this option if the default schema cannot be used. For example, use this option when the deployment login differs from the design-time login.</td>
<td>The value calculated by Dali.</td>
</tr>
<tr>
<td>JPA Implementation</td>
<td>Select to use the JPA implementation provided by the server at runtime, or select a specific implementation library that contain the Java Persistence API (JPA) and entities to be added to the project’s Java Build Path. Click <strong>Configure default JPA implementation library</strong> to create a default library for the project or click <strong>Configure user libraries</strong> to define additional libraries. Depending on your JPA implementation (for example, Generic or EclipseLink), different options may be available when working with JPA projects.</td>
<td>Determined by server.</td>
</tr>
<tr>
<td>Persistent class management</td>
<td>Specify if Dali will discover annotated classes automatically, or if the annotated classes must be listed in the persistence.xml file. <strong>Note:</strong> To insure application portability, you should explicitly list the managed persistence classes that are included in the persistence unit.</td>
<td>Determined by server.</td>
</tr>
<tr>
<td>Create orm.xml</td>
<td>Specify if Dali should create a default orm.xml file for your entity mappings and persistence unit defaults.</td>
<td>Selected</td>
</tr>
</tbody>
</table>
4.1.2.1 Entity Class page
This table lists the properties of the Entity Class page of the Create JPA Entity wizard.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>The name of the JPA project.</td>
<td></td>
</tr>
<tr>
<td>Source Folder</td>
<td>The location of the JPA project’s src folder.</td>
<td></td>
</tr>
<tr>
<td>Java Package</td>
<td>The name of the class package.</td>
<td></td>
</tr>
<tr>
<td>Class name</td>
<td>The name of the Java class.</td>
<td></td>
</tr>
<tr>
<td>Superclass</td>
<td>Select the superclass.</td>
<td></td>
</tr>
<tr>
<td>Inheritance</td>
<td>Because the wizard creates a Java class with an @Entity notation, the Entity option is selected by default.</td>
<td>Entity</td>
</tr>
<tr>
<td></td>
<td>Select Mapped Superclass if you defined a superclass.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>To add an @Inheritance notation to the entity, select Inheritance and then select one of the inheritance mapping strategies (described in JSR 220):</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- SINGLE_TABLE -- All classes in a hierarchy as mapped to a single table. This annotation is without an attribute for the inheritance strategy.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- TABLE_PER_CLASS -- Each class is mapped to a separate table.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- JOINED -- The root of the class hierarchy is represented by a single table. Each subclass is represented by a separate table that contains those fields that are specific to the subclass (not inherited from its superclass), as well as the column(s) that represent its primary key. The primary key column(s) of the subclass table serves as a foreign key to the primary key of the superclass table.</td>
<td></td>
</tr>
<tr>
<td>XML Entity Mappings</td>
<td>Select Add to entity mappings in XML to create XML mappings in orm.xml, rather than annotations. Use the Mapping file field to specify the file to use. By default, mappings are stored in the META-INF/orm.xml file.</td>
<td></td>
</tr>
</tbody>
</table>

4.1.2.2 Entity Properties page
This table lists the properties of the Entity Properties page of the Create JPA Entity wizard.

Table 4–1

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Name</td>
<td>The name of the entity. By default, this value is the same as the one entered as the class name. If the entity name differs from the class name, then the entity name is added as an attribute. For example: @Entity(name=&quot;EntityName&quot;).</td>
<td>Determined by server.</td>
</tr>
</tbody>
</table>
4.1.3 Mapping File Wizard

The Mapping File wizard enables you to add an orm.xml file to a JPA project if no object map exists at the location specified. For example, if you cleared the Create orm.xml option on the JPA Facet page, you can later add the orm.xml file to the src file of the project using this wizard.

The Mapping File Wizard consists of the Mapping File page.

4.1.3.1 Mapping File

This table lists the properties of the Mapping File Wizard.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table Name</td>
<td>Select Use default to match the name of the mapped table name to the entity name. Otherwise, clear the Use default option and enter the name in the Table Name field. These options result in the addition of the @Table option to the Java class file.</td>
<td>Use default.</td>
</tr>
<tr>
<td>Entity Fields</td>
<td>Click the Add button to add persistence fields using the Entity Fields dialog. This dialog enable you to build a field by entering a field name and selecting among persistence types. The Key option enables you to mark a field as a primary key. The dialog’s Browse function enables you to add other persistence types described in the JPA specification. The Edit button enables you to change the name or type set for a persistent field.</td>
<td></td>
</tr>
<tr>
<td>Access Type</td>
<td>Select whether the entity’s access to instance variables is field-based or property-based, as defined in the JPA specification.</td>
<td>Field</td>
</tr>
<tr>
<td></td>
<td>■ Field – Instance variables are accessed directly. All non-transient instance variables are persistent.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Property – Persistent state accessed through the property accessor methods. The property accessor methods must be public or private.</td>
<td></td>
</tr>
</tbody>
</table>

Table 4-1 (Cont.)
### 4.1.4 Generate Tables from Entities Wizard

Use the Generate DDL from Entities Wizard to quickly create DDL scripts from your persistent entities. Dali automatically creates the necessary primary and foreign keys, based on the entity mappings.

### 4.1.5 Generate Entities from Tables Wizard

Use the Generate Custom Entities Wizard to create entities from your database tables. The wizard consists of the following pages:

- Select Tables
- Table Associations
- Customize Default Entity Generation
- Customize Individual Entities

#### 4.1.5.1 Select Tables

Use the Select Tables dialog to specify the database connection and tables from which to create entities.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>Select a database connection or click Add Connection to create a new connection.</td>
</tr>
<tr>
<td>Schema</td>
<td>Select the database schema from which to select tables.</td>
</tr>
<tr>
<td>Tables</td>
<td>Select the tables from which to create Java persistent entities. The tables shown are determined by the database connection and schema selections.</td>
</tr>
</tbody>
</table>
4.1.5.2 Table Associations
Use this page to create or edit the association between the database table and entity.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table associations</td>
<td>Select an association to modify or click to create a new table association with the Create New Association wizard.</td>
</tr>
<tr>
<td>Generate this association</td>
<td>Specify if Dali should create the selected association. If enabled, you can specify the Cardinality and Join table for the table association.</td>
</tr>
<tr>
<td>Generate a reference to &lt;ROW&gt; in &lt;TABLE&gt;</td>
<td>Specify if the entity should contain a reference to the specified table. If enabled, you can also enter the Property name and select the Cascade method (all, persist, merge, remove, or refresh) for the reference.</td>
</tr>
</tbody>
</table>

4.1.5.3 Customize Default Entity Generation
Use this page to specify the default information Dali will use when generating the entities from the database tables. You will be able to override this information for specific entities.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table Mapping</td>
<td>Use these options to define the table mapping information for the entity.</td>
</tr>
<tr>
<td>Key generator</td>
<td>Select the generator used for this mapping.</td>
</tr>
<tr>
<td>Sequence name</td>
<td>Enter a name for the sequence. You can use $table and $pk as variables in the name. These will be replaced by the table name and primary key column name (respectively) when Dali generates a table mapping.</td>
</tr>
<tr>
<td>Entity access</td>
<td>Specify the default entity access method: Field (default) or Property.</td>
</tr>
<tr>
<td>Associations fetch</td>
<td>Specify the default fetch mode for associations: Default, as defined by the application (default), or Lazy.</td>
</tr>
<tr>
<td>Collection type</td>
<td>Specify if the collection properties are a Set or List. Enable the Always generate optional JPA annotations and DDL parameters option to have Dali include this information in the entity.</td>
</tr>
<tr>
<td>Domain Java Class</td>
<td>Use these options to define the class information (Package, Superclass, and Interfaces) for the entity.</td>
</tr>
</tbody>
</table>
4.1.5.4 Customize Individual Entities

Use this page to customize each generated entity. Select an item in the Table and columns area, then complete the following fields for each item.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table Mapping</td>
<td>Use these options to define the table mapping information for the entity.</td>
</tr>
<tr>
<td>Key generator</td>
<td>Select the generator used for this mapping.</td>
</tr>
<tr>
<td>Sequence name</td>
<td>Enter a name for the sequence.</td>
</tr>
<tr>
<td></td>
<td>You can use $\text{Table}$ and $\text{Spk}$ as variables in the name. These will be replaced by the table name and primary key column name (respectively) when Dali generates a table mapping.</td>
</tr>
<tr>
<td>Entity access</td>
<td>Specify the default entity access method: Field (default) or Property.</td>
</tr>
<tr>
<td>Domain Java Class</td>
<td>Use these options to define the class information (Superclass and Interfaces) for the entity.</td>
</tr>
</tbody>
</table>

4.1.6 Create New Association

Use the Create New Association wizard to specify association tables when generating an entity.

The wizard consists of the following pages:
- Association Tables
- Join Columns
- Association Cardinality

4.1.6.1 Association Tables

Use this page to specify the association tables for an entity.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Association kind</td>
<td>Specify if the association is Simple (1:M) or Many to Many (M:M).</td>
</tr>
<tr>
<td>Association tables</td>
<td>Click Table Selection, then select the two tables to associate. When creating a Many to Many association, you can select a Join Table for the association.</td>
</tr>
</tbody>
</table>

4.1.6.2 Join Columns

Use this dialog to specify the join columns of an association table.
Click Add to specify the join columns between the two tables.

### 4.1.6.3 Association Cardinality

Use this dialog to specify cardinality of an association table. Depending on the **Association Kind** and **Join Columns** that you selected previously, some associations may not be available.

- Many to one
- One to many
- One to one
- Many to many

### 4.2 Property pages

This section includes information on the following:

- JPA Details view (for entities)
- JPA Details view (for attributes)
- JPA Details view (for orm.xml)
- JPA Structure view

#### 4.2.1 JPA Details view (for entities)

The JPA Details view displays the persistence information for the currently selected entity and contains the following tabs:

- General information
- Attribute overrides
- Secondary table information
- Inheritance information

#### 4.2.1.1 General information

This table lists the General information fields available in the JPA Details view for each entity type.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Available for Entity Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping Type Hyperlink</td>
<td>Clicking the name of the mapping type, which is represented as a hyperlink, invokes the Mapping Type Selection dialog. Use this dialog to specify the type of entity: Mapped Superclass, Embeddable or the default mapping type.</td>
<td>Entity</td>
<td>Entity, Embeddable, and Mapped superclass</td>
</tr>
</tbody>
</table>
4.2.1.2 Attribute overrides

Use the Attribute Overrides area in the JPA Details view to override the default settings specified in the General information area of an attribute. Attribute overrides generally override/configure attributes that are inherited or embedded.

This table lists the Attribute override fields available in the JPA Details view for each entity type.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Available for Entity Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute Overrides</td>
<td>Specify a property or field to be overridden (from the default mappings). Select Override Default.</td>
<td>As defined in orm.xml.</td>
<td>Entity</td>
</tr>
<tr>
<td>Join Columns</td>
<td></td>
<td></td>
<td>Entity</td>
</tr>
</tbody>
</table>

4.2.1.3 Secondary table information

Use the Secondary Tables area in the JPA Details view to associate additional tables with an entity. Use this area if the data associated with an entity is spread across multiple tables.

Refer to "Specifying additional tables" on page 3-19 for additional information.

4.2.1.4 Inheritance information

This table lists the fields available on the Inheritance area in the JPA Details view for each entity type.
Refer to “Specifying entity inheritance” on page 3-19 for additional information.

### 4.2.1.5 Queries
Use the queries area of the JPA Details view to create named queries and named native queries. Refer to “Creating Named Queries” on page 3-21 for additional information.

### 4.2.2 JPA Details view (for attributes)

The JPA Details view displays the persistence information for the currently selected mapped attribute and contains the following areas:

- **General information**
- **Join Table Information**
- **Join Columns Information**
- **Primary Key Generation information**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy</td>
<td>Specify the strategy to use when mapping a class or class hierarchy:</td>
<td>Single table</td>
</tr>
<tr>
<td></td>
<td>- Single table – All classes in the hierarchy are mapped to a single table.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Joined – The root of the hierarchy is mapped to a single table; each child maps to its own table.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Table per class – Each class is mapped to a separate table.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This field corresponds to the @Inheritance annotation.</td>
<td></td>
</tr>
<tr>
<td>Discriminator Value</td>
<td>Specify the discriminator value used to differentiate an entity in this inheritance hierarchy. The value must conform to the specified Discriminator Type.</td>
<td></td>
</tr>
<tr>
<td>Discriminator Column</td>
<td>These fields are available when using a Single or Joined inheritance strategy.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This field corresponds to the @DiscriminatorColumn annotation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use the Details area to define the Length and Column definition of this Discriminator Column.</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Name of the discriminator column</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Set this field to set the discriminator type to Char or Integer (instead of its default: String). The Discriminator Value must conform to this type.</td>
<td>String</td>
</tr>
<tr>
<td>Primary Key Join Columns</td>
<td>Use to override the default primary key join columns. Select Override Default, then click Add to select new Join Column. This field corresponds with @PrimaryKeyJoinColumn annotation.</td>
<td></td>
</tr>
</tbody>
</table>
See "Mapping an entity" on page 3-22 for more information.

### 4.2.2.1 General information
This table lists the General properties available in the Java Details view for each mapping type.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Available for Mapping Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping Type Hyperlink</td>
<td>Clicking the name of the mapping type, which is represented as a hyperlink, invokes the Mapping Type Selection dialog. Use this dialog to specify the type of attribute.</td>
<td>Basic</td>
<td>All mapping types</td>
</tr>
<tr>
<td>Column</td>
<td>The database column that contains the value for the attribute. This field corresponds to the @Column annotation.</td>
<td>By default, the Column is assumed to be named identically to the attribute.</td>
<td>Basic mapping, Embedded mapping, ID mapping, Version mapping</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the database column. This field corresponds to the @Column annotation.</td>
<td></td>
<td>Basic mapping, Embedded mapping, ID mapping</td>
</tr>
<tr>
<td>Table</td>
<td>Name of the database table that contains the selected column.</td>
<td></td>
<td>Basic mapping, Embedded mapping, ID mapping</td>
</tr>
<tr>
<td>Insertable</td>
<td>Specifies if the column is always included in SQL INSERT statements.</td>
<td>True</td>
<td>Basic mapping, Embedded mapping, ID mapping</td>
</tr>
<tr>
<td>Updatable</td>
<td>Specifies if this column is always included in SQL UPDATE statements.</td>
<td>True</td>
<td>Basic mapping, Embedded mapping, ID mapping</td>
</tr>
<tr>
<td>Unique</td>
<td>Sets the UNIQUE constraint for the column.</td>
<td>False</td>
<td>Basic mapping, Embedded mapping, ID mapping</td>
</tr>
<tr>
<td>Nullable</td>
<td>Specifies if the column allows null values.</td>
<td>True</td>
<td>Basic mapping, Embedded mapping, ID mapping</td>
</tr>
<tr>
<td>Length</td>
<td>Sets the column length.</td>
<td>255</td>
<td>Basic mapping, Embedded mapping, ID mapping</td>
</tr>
<tr>
<td>Precision</td>
<td>Sets the precision for the column values.</td>
<td>0</td>
<td>Basic mapping, Embedded mapping, ID mapping</td>
</tr>
<tr>
<td>Scale</td>
<td>Sets the number of digits that appear to the right of the decimal point.</td>
<td>0</td>
<td>Basic mapping, Embedded mapping, ID mapping</td>
</tr>
<tr>
<td>Column Definition</td>
<td>Define the DDL for a column. This is used when a table is being generated.</td>
<td></td>
<td>Basic mapping, Embedded mapping, ID mapping</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default</td>
<td>Available for Mapping Type</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td>Fetch Type</td>
<td>Defines how data is loaded from the database:</td>
<td>Eager</td>
<td>Basic mapping, One-to-one mapping, Many-to-many mapping, Many-to-one mapping</td>
</tr>
<tr>
<td></td>
<td>■ Eager – Data is loaded in before it is actually needed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Lazy – Data is loaded only when required by the transaction.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>Specifies if this field is can be null.</td>
<td>Yes</td>
<td>Basic mapping, One-to-one mapping, Many-to-one mapping</td>
</tr>
<tr>
<td>Lob</td>
<td>Specify if the field is mapped to java.sql.Clob or java.sql.Blob. This field corresponds to the @Lob annotation.</td>
<td></td>
<td>Basic mapping</td>
</tr>
<tr>
<td>Temporal</td>
<td>Specifies if this field is one of the following:</td>
<td></td>
<td>Basic mapping, ID mapping</td>
</tr>
<tr>
<td></td>
<td>■ Date – java.sql.Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Time – java.sql.Time</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Timestamp – java.sql.Timestamp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enumerated</td>
<td>Specify how to persist enumerated constraints if the String value suits your application requirements or to match an existing database schema.</td>
<td>Ordinal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ ordinal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ String</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target Entity</td>
<td>The persistent entity to which the attribute is mapped.</td>
<td></td>
<td>One-to-one mapping, One-to-many mapping, Many-to-many mapping, Many-to-one mapping</td>
</tr>
<tr>
<td>Cascade Type</td>
<td>Specify which operations are propagated throughout the entity.</td>
<td></td>
<td>One-to-one mapping, One-to-many mapping, Many-to-many mapping, Many-to-one mapping</td>
</tr>
<tr>
<td></td>
<td>■ All – All operations</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Persist</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Merge</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Move</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Remove</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Refresh</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 4.2.2.2 Join Table Information

Use area to specify a mapped column for joining an entity association. By default, the mapping is assumed to have a single join.

This table lists the fields available on the Join Table area in the JPA Details view for **One-to-many mapping** and **Many-to-many mapping** mapping types.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Available for Mapping Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped By</td>
<td>The field in the database table that &quot;owns&quot; the relationship. This field is required only on the non-owning side of the relationship.</td>
<td></td>
<td>One-to-one mapping, One-to-many mapping</td>
</tr>
<tr>
<td>Order By</td>
<td>Specify the default order for objects returned from a query:</td>
<td>Primary key</td>
<td>One-to-many mapping, Many-to-many mapping, Many-to-one mapping</td>
</tr>
<tr>
<td></td>
<td>- No ordering</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Primary key</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Custom ordering</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This field corresponds to the @OrderBy annotation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute Overrides</td>
<td>Overrides Basic mappings of a mapped superclass (for example, if the inherited column name is incompatible with a pre-existing data model, or invalid as a column name in your database).</td>
<td></td>
<td>Embedded mapping</td>
</tr>
</tbody>
</table>

**4.2.2.3 Join Columns Information**

This table lists the fields available in the Join Table area in JPA Details view for **Many-to-one mapping** and **One-to-one mapping** mapping types.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the join table that contains the foreign key column.</td>
<td>By default, the name is assumed to be the primary tables associated with the entities concatenated with an underscore.</td>
</tr>
<tr>
<td>Join Columns</td>
<td>Specify a mapped column for joining an entity association. This field corresponds to the @JoinColumn attribute. Select <strong>Override Default</strong>, then Add, Edit, or Remove the join columns.</td>
<td>By default, the mapping is assumed to have a single join.</td>
</tr>
<tr>
<td>Inverse Join Columns</td>
<td>Select <strong>Override Default</strong>, then Add, Edit, or Remove the join columns.</td>
<td></td>
</tr>
</tbody>
</table>
4.2.2.4 Primary Key Generation information

This table lists the fields available in the Primary Key Generation area in JPA Details view for ID mapping types.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join Column</td>
<td>Specify a mapped column for joining an entity association. This field corresponds to the @JoinColumn attribute. Select Override Default, then Add, Edit, or Remove the join columns.</td>
<td>By default, the mapping is assumed to have a single join.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Key Generation</td>
<td>These fields define how the primary key is generated. These fields correspond to the @GeneratedValue annotation.</td>
<td>Generated Value</td>
</tr>
<tr>
<td>Strategy</td>
<td>- Auto&lt;br&gt;- Identity – Values are assigned by the database’s Identity column.&lt;br&gt;- Sequence – Values are assigned by a sequence table (see Sequence Generator).&lt;br&gt;- Table – Values are assigned by a database table (see Table Generator).</td>
<td>Auto</td>
</tr>
<tr>
<td>Generator Name</td>
<td>Unique name of the generated value.</td>
<td></td>
</tr>
<tr>
<td>Table Generator</td>
<td>These fields define the database table used for generating the primary key and correspond to the @TableGenerator annotation. These fields apply only when Strategy = Table.</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Unique name of the generator.</td>
<td></td>
</tr>
<tr>
<td>Table</td>
<td>Database table that stores the generated ID values.</td>
<td></td>
</tr>
<tr>
<td>Primary Key Column</td>
<td>The column in the table generator’s Table that contains the primary key.</td>
<td></td>
</tr>
<tr>
<td>Value Column</td>
<td>The column that stores the generated ID values.</td>
<td></td>
</tr>
<tr>
<td>Primary Key Column Value</td>
<td>The value for the Primary Key Column in the generator table.</td>
<td></td>
</tr>
</tbody>
</table>
4.2.3 JPA Details view (for orm.xml)

The JPA Details view displays the default mapping and persistence information for the project and contains the following areas:

- General information
- Persistence Unit information
- Generators
- Queries
- Converters (when using EclipseLink)

These defaults can be overridden by the settings on a specific entity or mapping.

### 4.2.3.1 General information

This table lists the General information fields available in the JPA Details view for each entity type.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Generator</td>
<td>These fields define the specific sequence used for generating the primary key and correspond to the @SequenceGenerator annotation. These fields apply only when Strategy = Sequence.</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Name of the sequence table to use for defining primary key values.</td>
<td></td>
</tr>
<tr>
<td>Sequence</td>
<td>Unique name of the sequence.</td>
<td></td>
</tr>
<tr>
<td>Package</td>
<td>The Java package that contains the persistent entities. Click Browse and select the package.</td>
<td></td>
</tr>
<tr>
<td>Schema</td>
<td>The database schema that contains the Table. This field corresponds to the &lt;schema&gt; element in the orm.xml file.</td>
<td></td>
</tr>
<tr>
<td>Catalog</td>
<td>The database catalog that contains the Table. This field corresponds to the &lt;catalog&gt; element in the orm.xml file.</td>
<td></td>
</tr>
<tr>
<td>Access</td>
<td>Specify the default access method for the variables in the project:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Property</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This field corresponds to the &lt;access&gt; element in the orm.xml file.</td>
<td></td>
</tr>
</tbody>
</table>
4.2.3.2 Persistence Unit information

This table lists the Persistence Unit information fields available in the JPA Details view for each entity type. These fields are contained in the `<persistence-unit-metadata>` element in the `orm.xml` file.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Mapping Data Complete</td>
<td>Specifies that the Java classes in this persistence unit are fully specified by their metadata. Any annotations will be ignored. This field corresponds to the <code>&lt;xml-mapping-metadata-complete&gt;</code> element in the <code>orm.xml</code> file.</td>
<td></td>
</tr>
<tr>
<td>Schema</td>
<td>The database schema that contains the <strong>Table</strong>. This field corresponds to the <code>&lt;schema&gt;</code> element in the <code>orm.xml</code> file.</td>
<td></td>
</tr>
<tr>
<td>Catalog</td>
<td>The database catalog that contains the <strong>Table</strong>. This field corresponds to the <code>&lt;catalog&gt;</code> element in the <code>orm.xml</code> file.</td>
<td></td>
</tr>
<tr>
<td>Access</td>
<td>Specify how the entity its access instance variables.</td>
<td>Property</td>
</tr>
<tr>
<td></td>
<td>- Property – Persistent state accessed through the property accessor methods. The property accessor methods must be <strong>public</strong> or <strong>private</strong>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Field – Instance variables are accessed directly. All non-transient instance variables are persistent.</td>
<td></td>
</tr>
<tr>
<td>Cascade Persist</td>
<td>Adds cascade-persist to the set of cascade options in entity relationships of the persistence unit. This field corresponds to the <code>&lt;cascade-persist&gt;</code> element in the <code>orm.xml</code> file.</td>
<td></td>
</tr>
</tbody>
</table>

4.2.3.3 Generators

This table lists the Generator information fields available in the JPA Details view for the `orm.xml` file.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| Generator| Displays the existing Sequence and Table generators. Click **Add Sequence** or **Add Table** to add a new generator. For sequence generators, you must complete the following fields:  
  - Name  
  - Sequence  
  For table generators, you must complete the following fields:  
  - Name  
  - Table  
  - Primary key column  
  - Value column  
  - Primary key column value |
4.2.3.4 Queries
This table lists the Query information fields available in the JPA Details view for the `orm.xml` file.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries</td>
<td>Displays the existing Named and Native queries. Click <strong>Add</strong> to add a named query, or <strong>Add Native</strong> for a native query. For named queries, enter the query in the Query field. For native queries, select a result class, then enter the query in the Query field.</td>
</tr>
<tr>
<td>Query Hints</td>
<td>Displays the existing query hints (Name and Value). Click <strong>Add</strong> to add a new query hint.</td>
</tr>
</tbody>
</table>

4.2.3.5 Converters
The Converters information in the JPA Details view applies only when using EclipseLink. Click **Add** to create a new converter, using the **Add Converter** dialog.

4.2.4 JPA Structure view
The JPA Structure view displays an outline of the structure (its attributes and mappings) of the entity that is currently selected or opened in the editor. The structural elements shown in the outline are the entity and its fields.

**Figure 4–1  Sample JPA Structure View**

4.2.5 persistence.xml Editor
The persistence.xml Editor provides an interface that enables you to update the persistence.xml file. For projects using the EclipseLink platform, the persistence.xml Editor consists of the following pages:

- General
- Connection
- Customization
- Caching
For projects using the Generic platform, the following subset of these pages is available:

- General
- Connection
- Properties
- Source

### 4.2.5.1 General

The following table lists properties available in the General page of the persistence.xml Editor.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter the name of the persistence unit.</td>
<td>The project name.</td>
</tr>
<tr>
<td>Persistence Provider</td>
<td>Enter the name of the persistence provider.</td>
<td>Determined by the server.</td>
</tr>
<tr>
<td>Description</td>
<td>Enter a description for this persistence unit. This is an optional property.</td>
<td></td>
</tr>
<tr>
<td>Managed Classes</td>
<td>Add or remove the classes managed through the persistence unit.</td>
<td></td>
</tr>
<tr>
<td>Exclude Unlisted Classes</td>
<td>Select to include all annotated entity classes in the root of the persistence unit.</td>
<td>False</td>
</tr>
<tr>
<td>XML Mapping Files</td>
<td>Add or remove the object/relational mapping XML files that define the classes to be managed by the persistence unit.</td>
<td>Meta-INF\orm.xml</td>
</tr>
</tbody>
</table>

### 4.2.5.2 Connection

The following table lists the properties available in the Connection page of the persistence.xml Editor.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction Type</td>
<td>Specify if the connection for this persistence unit uses one of the following transaction types:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- <strong>Default</strong> -- Select to use the container used by the container.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- <strong>JTA</strong> (Java Transaction API) -- Transactions of the Java EE server.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- <strong>Resource Local</strong> -- Native actions of a JDBC driver that are referenced by a persistence unit.</td>
<td></td>
</tr>
</tbody>
</table>
### Properties of the Connection Page

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch Writing</td>
<td>Specify the use of batch writing to optimize transactions with multiple write operations. Set the value of this property into the session at deployment time. Note: This property applies when used both in a Java SE and Java EE environment. The following are the valid values for oracle.toplink.config.BatchWriting:</td>
<td>None</td>
</tr>
</tbody>
</table>
|                                 | - **JDBC**—Use JDBC batch writing.  
|                                 | - **Buffered**—Do not use either JDBC batch writing nor native platform batch writing.  
|                                 | - **OracleJDBC**—Use both JDBC batch writing and Oracle native platform batch writing.  
|                                 | - **None**—Do not use batch writing (turn it off).                                                                                                                                                           |         |
| Statement caching               |                                                                                                                                                                                                             |         |
| Native SQL                      | If you selected **JTA** as the transaction type, then enter the name of the default JTA data source for the persistence unit.                                                                                   | False   |
| JTA Data Source Name            | If you selected **JTA** as the transaction type, then enter the name of the default JTA data source for the persistence unit.                                                                               |         |
| Non-JTA Data Source Name        | If you selected **Resource Local** as the transaction type, then enter the name of the non-JTA data source.                                                                                                   |         |
|                                 | This property is not available for projects using the Generic platform.                                                                                                                                       |         |
| Bind Parameters                 | Control whether or not the query uses parameter binding.                                                                                                                                                     |         |
|                                 | Note: This property applies when used in a Java SE environment.                                                                                                                                              |         |
|                                 | This property is not available for projects using the Generic platform.                                                                                                                                       |         |
| EclipseLink Connection Pool     | Define the connection pool driver, URL, user name and password.                                                                                                                                              |         |
|                                 | These properties are not available for projects using the Generic platform.                                                                                                                                    |         |
| Read Connection                 | The maximum and minimum number of connections allowed in the JDBC read connection pool.                                                                                                                                              |         |
|                                 | Note: These properties apply when used in a Java SE environment.                                                                                                                                              |         |
|                                 | These properties are not available for projects using the Generic platform.                                                                                                                                   |         |
| Write Connection                | The maximum and minimum number of connections allowed in the JDBC read connection pool.                                                                                                                                              |         |
|                                 | Note: These properties apply when used in a Java SE environment.                                                                                                                                              |         |
|                                 | These properties are not available for projects using the Generic platform.                                                                                                                                    |         |
4.2.5.3 Customization

The following table lists the properties available in the Customization page of the persistence.xml Editor.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| Weaving                | Specifies if weaving of the entity classes is performed. The EclipseLink JPA persistence provider uses weaving to enhance JPA entities for such properties as lazy loading, change tracking, fetch groups, and internal optimizations. Select from the following options:  
  ■ No Weaving  
  ■ Weave Dynamically  
  ■ Weave Statically -- Use this option if you plan to execute your application outside of a Java EE 5 container in an environment that does not permit the use of -javaagent:eclipselink.jar on the JVM command line. This assumes that classes have already been statically woven. Run the static weaver on the classes before deploying them. | Weave Dynamically              |
| Weaving Lazy           | Select this option to enable lazy weaving.                                                                                                                                                                                                                                  | True                          |
| Weaving Fetch Groups   | Select this option to enable fetch groups through weaving. Set this option to false if:  
  ■ There is no weaving.  
  ■ Classes should not be changed during weaving (for example, when debugging).  
  Set this property to false for platforms where it is not supported.                                                                                                                                                  | True                          |
| Weaving internal       |                                                                                                                                                                                                                                                                            | True                          |
| Weaving eager          |                                                                                                                                                                                                                                                                            | False                         |
| Weaving Change Tracking| Select this option to use weaving to detect which fields or properties of the object change.                                                                                                                                                                                   | True                          |
| Throw Exceptions       | Select this option to set EclipseLink to throw an exception or log a warning when it encounters a problem with any of the files listed in a persistence.xml file <mapping-file> element.                                                                                   | True                          |
| Exception handler      | Select (or create) a Java class to handle exceptions.                                                                                                                                                                                                                      |                               |
| Session Customizer     | Select a session customizer class: a Java class that implements the eclipselink.tools.sessionconfiguration.SessionCustomizer interface and provides a default (zero-argument) constructor. Use this class’ customize method, which takes an eclipselink.sessions.Session to programmatically access advanced EclipseLink session API. |                               |
Table 4–5 (Cont.) Properties of the Customization Page

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Descriptor Customizer</td>
<td>Select an EclipseLink descriptor customizer class—a Java class that implements the <code>eclipselink.tools.sessionconfiguration.DescriptorCustomizer</code> interface and provides a default (zero-argument) constructor. Use this class's <code>customize</code> method, which takes an <code>eclipselink.descriptors.ClassDescriptor</code>, to programmatically access advanced EclipseLink descriptor and mapping API for the descriptor associated with the JPA entity named <code>&lt;ENTITY&gt;</code>.</td>
<td></td>
</tr>
<tr>
<td>Validation only</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>Profiler</td>
<td>NoProfiler, Performance Profiler, Query Monitor</td>
<td>NoProfiler</td>
</tr>
</tbody>
</table>

**Note:** This page is not available for projects using the Generic platform.

### 4.2.5.4 Caching

This table lists the properties of the Caching page of the `persistence.xml` Editor.
Table 4–6  Properties of the Caching Page

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Cache Type</td>
<td>Select one of the following as the Default Cache Type:</td>
<td>Weak with hard subcache</td>
</tr>
<tr>
<td>■ Soft with Weak Subcache</td>
<td>This option is similar to <strong>Weak with Hard Subcache</strong> except that it maintains a most frequently used subcache that uses soft references. The size of the subcache is proportional to the size of the identity map. The subcache uses soft references to ensure that these objects are garbage-collected only if the system is low on memory. Use this identity map in most circumstances as a means to control memory used by the cache.</td>
<td></td>
</tr>
<tr>
<td>■ Weak with Hard Subcache</td>
<td>This option is similar to <strong>Soft with Weak subcache</strong> except that it maintains a most frequently used subcache that uses hard references. Use this identity map if soft references are not suitable for your platform.</td>
<td></td>
</tr>
<tr>
<td>■ Weak</td>
<td>This option is similar to <strong>Full</strong>, except that objects are referenced using weak references. This option uses less memory than <strong>Full</strong>, allows complete garbage collection and provides full caching and guaranteed identity. Use this identity map for transactions that, once started, stay on the server side.</td>
<td></td>
</tr>
<tr>
<td>■ Soft</td>
<td>This option is similar to <strong>Weak</strong> except that the map holds the objects using soft references. This identity map enables full garbage collection when memory is low. It provides full caching and guaranteed identity.</td>
<td></td>
</tr>
<tr>
<td>■ Full</td>
<td>This option provides full caching and guaranteed identity: all objects are cached and not removed. Note: This process may be memory-intensive when many objects are read.</td>
<td></td>
</tr>
<tr>
<td>■ None</td>
<td>This option does not preserve object identity and does not cache objects. This option is not recommended.</td>
<td></td>
</tr>
<tr>
<td>Default Cache Size</td>
<td>Set the size (maximum number of objects) of the cache.</td>
<td>100</td>
</tr>
<tr>
<td>Default Shared Cache</td>
<td>Specifies if cached instances should be in the shared cache or in a client isolated cache.</td>
<td>True</td>
</tr>
<tr>
<td>Entity Caching</td>
<td>Specify the entity-specific caching information.</td>
<td></td>
</tr>
<tr>
<td>Cache Type</td>
<td>See <strong>Default Cache Type</strong>.</td>
<td></td>
</tr>
<tr>
<td>Cache Size</td>
<td>See <strong>Default Cache Size</strong>.</td>
<td></td>
</tr>
<tr>
<td>Shared Cache</td>
<td>See <strong>Default Shared Cache</strong>.</td>
<td></td>
</tr>
</tbody>
</table>
### 4.2.5.5 Logging

This table lists the properties of the Logging page of the `persistence.xml` Editor.

#### Note: This page is not available for projects using the Generic platform.

---

#### Table 4–6 (Cont.) Properties of the Caching Page

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flush clear cache</td>
<td>Select one of the following as the Default Cache Type:</td>
<td>Drop Invalidate</td>
</tr>
<tr>
<td></td>
<td>■ <strong>Drop</strong> – This mode is the fastest and uses the least memory. However, after commit the shared cache might potentially contain stale data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ <strong>Drop Invalidate</strong> – Classes that have at least one object updated or deleted are invalidated in the shared cache at commit time. This mode is slower than <strong>Drop</strong>, but as efficient memory usage-wise, and prevents stale data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ <strong>Merge</strong> – Drop classes from the EntityManager’s cache of objects that have not been flushed. This mode leaves the shared cache in a perfect state after commit. However, it is the least memory-efficient mode; the memory might even run out in a very large transaction.</td>
<td></td>
</tr>
</tbody>
</table>

---

#### Note: This page is not available for projects using the Generic platform.
Table 4–7 Properties of the Logging Page

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logging Level</td>
<td>Specifies the amount and detail of log output by selecting the log level (in ascending order of information):</td>
<td>Info</td>
</tr>
<tr>
<td></td>
<td>The following are the valid values for the java.util.logging.Level:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ OFF—disables logging</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ SEVERE—logs exceptions indicating TopLink cannot continue, as well as any exceptions generated during login. This includes a stack trace.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ WARNING—logs exceptions that do not force TopLink to stop, including all exceptions not logged with severe level. This does not include a stack trace.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ INFO—logs the login/logout per sever session, including the user name. After acquiring the session, detailed information is logged.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ CONFIG—logs only login, JDBC connection, and database information.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ FINE—logs SQL.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ FinER—similar to warning. Includes stack trace.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ FinEST—includes additional low level information.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example: persistence.xml file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;property</td>
<td></td>
</tr>
<tr>
<td></td>
<td>name=&quot;eclipselink.logging.level&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>value=&quot;INFO&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timestamp</td>
<td>Control whether the timestamp is logged in each log entry.</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>The following are the valid values:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ true—log a timestamp.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ false—do not log a timestamp.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example: persistence.xml file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;property</td>
<td></td>
</tr>
<tr>
<td></td>
<td>name=&quot;eclipselink.logging.timestamp&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>value=&quot;false&quot;</td>
<td></td>
</tr>
<tr>
<td>Thread</td>
<td>Control whether a thread identifier is logged in each log entry.</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>The following are the valid values:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ true—log a thread identifier.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ false—do not log a thread identifier.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| Session    | Control whether an EclipseLink session identifier is logged in each log entry. The following are the valid values:  
  - true–log a EclipseLink session identifier.  
  - false–do not log a EclipseLink session identifier.  
  Example: persistence.xml file  
    ```xml  
    <property  
      name="eclipselink.logging.session"  
      value="false"/>  
    ``` | true    |
| Exceptions | Control whether the exceptions thrown from within the EclipseLink code are logged prior to returning the exception to the calling application. Ensures that all exceptions are logged and not masked by the application code.  
  The following are the valid values:  
  - true–log all exceptions.  
  - false–do not log exceptions.  
  Example: persistence.xml file  
    ```xml  
    <property  
      name="eclipselink.logging.exceptions"  
      value="true"/>  
    ``` | false   |
| Log file   | Specify a file location for the log output (instead of the standard out).  
  Example: persistence.xml file  
    ```xml  
    <property name="eclipselink.logging.file"  
      value="C:\myout"/>  
    ``` |         |
### 4.2.5.6 Options

This table lists the properties of the Options page of the persistence.xml Editor.

#### Note: This page is not available for projects using the Generic platform.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logger</td>
<td>Select the type of logger to use: The following are the valid values:</td>
<td>DefaultLogger</td>
</tr>
<tr>
<td></td>
<td>■ DefaultLogger—the EclipseLink native logger</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eclipselink.logging.DefaultSessionLog.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ JavaLogger—the java.util.logging logger</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eclipselink.logging.JavaLog.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ ServerLogger—the java.util.logging logger</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eclipselink.platform.server.ServerLog.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Integrates with the application server's logging as defined in the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eclipselink.platform.server.ServerPlatform.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Fully qualified class name of a custom logger. The custom logger must</td>
<td></td>
</tr>
<tr>
<td></td>
<td>implement the eclipselink.logging.SessionLog interface.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example: persistence.xml file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;property name=&quot;eclipselink.logging.logger&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>value=&quot;acme.loggers.MyCustomLogger&quot; /&gt;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session Name</td>
<td>Specify the name by which the EclipseLink session is stored in the static</td>
<td></td>
</tr>
<tr>
<td></td>
<td>session manager. Use this option if you need to access the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EclipseLink shared session outside of the context of the JPA or to use</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a pre-existing EclipseLink session configured through a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EclipseLink sessions.xml file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Valid values: a valid EclipseLink session name that is unique in a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>server deployment.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example: persistence.xml file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;property name=&quot;eclipselink.session-name&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>value=&quot;MySession&quot; /&gt;</td>
<td></td>
</tr>
</tbody>
</table>
### Sessions XML

Specify persistence information loaded from the EclipseLink session configuration file (`sessions.xml`).

You can use this option as an alternative to annotations and deployment XML. If you specify this property, EclipseLink will override all class annotation and the object relational mapping from the `persistence.xml`, as well as `ORM.xml` and other mapping files, if present.

Indicate the session by setting the `eclipselink.session-name` property.

**Note:** If you do not specify the value for this property, `sessions.xml` file will not be used.

Valid values: the resource name of the sessions XML file.

**Example:** `persistence.xml` file

```xml
<property name="toqlink.session-xml" value="mysession.xml"/>
```

### Target Database

Select the target database. You can also set the value to the fully qualified classname of a subclass of the `org.eclipse.persistence.platform.DatabasePlatform` class.

**Example:** `persistence.xml` file

```xml
<property name="eclipselink.target-database" value="Oracle"/>
```

### Target Server

Select the target server for your JPA application.

**Example:** `persistence.xml` file

```xml
<property name="eclipselink.target-server" value="OC4J_10_1_3"/>
```

### Event Listener

Specify a descriptor event listener to be added during bootstrapping.

Valid values: qualified class name for a class that implements the `eclipselink.sessions.SessionEventListener` interface.

**Example:** `persistence.xml` file

```xml
<property name="eclipselink.session-event-listener" value="mypackage.MyClass.class"/>
```

### Include Descriptor Queries

Enable or disable the default copying of all named queries from the descriptors to the session. These queries include the ones defined using EclipseLink API, descriptor amendment methods, and so on.

### Miscellaneous Options

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sessions XML</td>
<td>Specify persistence information loaded from the EclipseLink session configuration file (<code>sessions.xml</code>). You can use this option as an alternative to annotations and deployment XML. If you specify this property, EclipseLink will override all class annotation and the object relational mapping from the <code>persistence.xml</code>, as well as <code>ORM.xml</code> and other mapping files, if present. Indicate the session by setting the <code>eclipselink.session-name</code> property. <strong>Note:</strong> If you do not specify the value for this property, <code>sessions.xml</code> file will not be used. Valid values: the resource name of the sessions XML file. <strong>Example:</strong> <code>persistence.xml</code> file</td>
<td></td>
</tr>
<tr>
<td>Target Database</td>
<td>Select the target database. You can also set the value to the fully qualified classname of a subclass of the <code>org.eclipse.persistence.platform.DatabasePlatform</code> class. <strong>Example:</strong> <code>persistence.xml</code> file</td>
<td>Auto</td>
</tr>
<tr>
<td>Target Server</td>
<td>Select the target server for your JPA application. <strong>Example:</strong> <code>persistence.xml</code> file</td>
<td>None</td>
</tr>
<tr>
<td>Event Listener</td>
<td>Specify a descriptor event listener to be added during bootstrapping. Valid values: qualified class name for a class that implements the <code>eclipselink.sessions.SessionEventListener</code> interface. <strong>Example:</strong> <code>persistence.xml</code> file</td>
<td></td>
</tr>
<tr>
<td>Include Descriptor Queries</td>
<td>Enable or disable the default copying of all named queries from the descriptors to the session. These queries include the ones defined using EclipseLink API, descriptor amendment methods, and so on.</td>
<td></td>
</tr>
</tbody>
</table>
4.2.5.7 Schema Generation

This table lists the properties of the Schema Generation page of the *persistence.xml* Editor.

**Note:** This page is not available for projects using the Generic platform.

### Table 4–8 (Cont.) Properties of the Options Page

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temporal mutable</td>
<td>Specify if all Date and Calendar persistent fields should be handled as mutable objects.</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong> <em>persistence.xml</em> file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;property name=&quot;eclipselink.temporal.mutable&quot; value=&quot;true&quot;/&gt;</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4–9

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDL Generation Type</td>
<td>Select the type of DDL generation:</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>■ None -- Do not generate DDL; no schema is generated.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Create Tables -- Create DDL for non-existent tables; leave existing tables unchanged.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Drop and Create Tables -- Create DDL for all tables; drop all existing tables.</td>
<td></td>
</tr>
<tr>
<td>Output Mode</td>
<td>Select the DDL generation target:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Both -- Generate SQL files and execute them on the database.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Database -- Execute SQL on the database only (do not generate SQL files).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ SQL Script -- Generate SQL files only (do not execute them on the database).</td>
<td></td>
</tr>
<tr>
<td>DDL Generation Location</td>
<td>Specify where EclipseLink writes DDL output. Specify a file specification to a directory in which you have write access. The file specification may be relative to your current working directory or absolute. If it does not end in a file separator, then EclipseLink appends one that is valid for your operating system.</td>
<td></td>
</tr>
<tr>
<td>Create DDL File Name</td>
<td>Specify the file name of the DDL file that EclipseLink generates that contains SQL statements for creating tables for JPA entities. Specify a file name valid for your operating system.</td>
<td>createDDL.jdbc</td>
</tr>
<tr>
<td>Drop DDL File Name</td>
<td>Specify the file name of the DDL file that EclipseLink generates that contains SQL statements for dropping tables for JPA entities.</td>
<td>dropDDL.jdbc</td>
</tr>
</tbody>
</table>
4.2.5.8 Properties
This page enables you to add or remove the vendor-specific <properties> elements of persistence.xml.

To add a property, click Add then enter the property Name and Value.

4.2.5.9 Source
Using this page, you can manually edit the persistence.xml file.
See "Managing the persistence.xml file" on page 3-10 for additional information.

4.3 Preferences
This section includes information on the following preference pages:

- Project Properties page – Java Persistence Options

4.3.1 Project Properties page – Java Persistence Options
Use the Java Persistence options on the Properties page to select the database connection to use with the project.

This table lists the properties available in the JPA Details page.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>Select the vendor-specific platform.</td>
</tr>
<tr>
<td>Library</td>
<td>Select a specific JPA library configuration.</td>
</tr>
<tr>
<td></td>
<td>Click Manage libraries to create or update a user library.</td>
</tr>
<tr>
<td></td>
<td>Click Download libraries to download a specific library configuration.</td>
</tr>
<tr>
<td>Type</td>
<td>Select User Library to select from the available user-defined or downloaded libraries.</td>
</tr>
<tr>
<td></td>
<td>If you select Disable, you must manually include the JPA implementation library on the project classpath.</td>
</tr>
<tr>
<td>Include libraries with this application</td>
<td>Specify if the selected libraries are included when deploying the application.</td>
</tr>
<tr>
<td>Connection</td>
<td>The database connection used to map the persistent entities.</td>
</tr>
<tr>
<td></td>
<td>- To create a new connection, click Add Connections.</td>
</tr>
<tr>
<td></td>
<td>- To reconnect to an existing connection, click Reconnect.</td>
</tr>
<tr>
<td>Override default catalog from connection</td>
<td>Select a catalog other than the default one derived from the connection information. Use this option if the default catalog is incorrect or cannot be used.</td>
</tr>
<tr>
<td>Override default schema from connection</td>
<td>Select a schema other than the default one derived from the connection information. Use this option if the default schema is incorrect or cannot be used. For example, use this option when the deployment login differs from the design-time login.</td>
</tr>
</tbody>
</table>
4.4 Dialogs

This section includes information on the following preference pages:

- Edit Join Columns Dialog
- Select Cascade dialog
- New EclipseLink Mapping File dialog

4.4.1 Edit Join Columns Dialog

Use the Join Columns dialog to create or modify the join tables and columns in relationship mappings.

This table lists the properties available in the Join Columns dialog.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistent Class Management</td>
<td>Specify if Dali will discover annotated classes automatically, or if the annotated classes must be listed in the persistence.xml file. Note: To insure application portability, you should explicitly list the managed persistence classes that are included in the persistence unit.</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the joint table column that contains the foreign key column.</td>
</tr>
<tr>
<td>Referenced Column Name</td>
<td>Name of the database column that contains the foreign key reference for the entity relationship.</td>
</tr>
</tbody>
</table>

4.4.2 Select Cascade dialog

Specify which operations are propagated throughout the association: All, Persist, Merge, Remove, or Refresh.

4.4.3 New EclipseLink Mapping File dialog

Specify the location and properties of the EclipseLink mapping file (eclipselink-orm.xml).

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>Select the project in which to add the mapping file.</td>
</tr>
<tr>
<td>Source folder</td>
<td>Click Browse and select the source file in which to add the mapping file. The default is ../&lt;PROJECT&gt;/src.</td>
</tr>
</tbody>
</table>
4.4.4 Add Converter dialog

Use this dialog to create a new EclipseLink converter.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter the name for this converter</td>
</tr>
<tr>
<td>Type</td>
<td>Select the converter type:</td>
</tr>
<tr>
<td></td>
<td>■ Custom</td>
</tr>
<tr>
<td></td>
<td>■ Object type</td>
</tr>
<tr>
<td></td>
<td>■ Struct</td>
</tr>
<tr>
<td></td>
<td>■ Type</td>
</tr>
</tbody>
</table>

4.5 JPA Development perspective

The JPA Development perspective defines the initial set and layout of views in the Workbench window when using Dali. By default, the JPA Development perspective includes the following views:

■ JPA Structure view
■ JPA Details view (for entities)
■ JPA Details view (for attributes)
■ JPA Details view (for orm.xml)

4.6 Icons and buttons

This section includes information on each of the icons and buttons used in the Dali OR Mapping Tool.

■ Icons
■ Buttons

4.6.1 Icons

The following icons are used throughout the Dali OR Mapping Tool.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Entity icon]</td>
<td>Entity</td>
</tr>
</tbody>
</table>
4.6.2 Buttons

The following buttons are used throughout the Dali OR Mapping Tool.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Embeddable entity</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Mapped superclass</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Basic mapping</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Basic collection mapping</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Basic map mapping</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Embedded mapping</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Embedded ID mapping</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>ID mapping</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Many-to-many mapping</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Many-to-one mapping</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>One-to-many mapping</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>One-to-one mapping</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Transformation mappings</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Transient mapping</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Variable one-to-one mappings</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Version mapping</td>
</tr>
</tbody>
</table>

4.7 Dali Developer Documentation

Additional Dali documentation is available online at:


This developer documentation includes information about:
- Dali architecture
- Plugins that comprise the Dali JPA Eclipse feature
- Extension points
The following tips and tricks give some helpful ideas for increasing your productivity.

- **Database Connections**
- **Schema-based persistence.xml**

<table>
<thead>
<tr>
<th>Tip</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Database Connections</strong></td>
<td>When starting a new workbench session, be sure to reconnect to your database (if you are working online). This allows Dali to provide database-related mapping assistance and validation.</td>
</tr>
<tr>
<td><strong>Schema-based persistence.xml</strong></td>
<td>If you are behind a firewall, you may need to configure your Eclipse workspace proxy in the Preferences dialog (Preferences &gt; Internet &gt; Proxy Settings) to properly validate a schema-based persistence.xml file.</td>
</tr>
</tbody>
</table>
This section contains descriptions of the following new features and significant changes made to the Dali OR Mapping Tool for Release 2.2:

- JPA Library Management
- New Generate Entities from Tables Wizard
- Conversion of Java Projects to JPA Projects
- EclipseLink 1.1 Support

### 6.1 JPA Library Management

Use the new [JPA Facet page](#) to create and manage JPA libraries when creating a new JPA project. You can specify a JPA library configuration from a selected JPA implementation, or create a custom library.

The Dali OR Mapping Tool will automatically validate the library to ensure that the correct classes are available for a given platform.

### 6.2 New Generate Entities from Tables Wizard

Use the new [Generate Entities from Tables Wizard](#) to create JPA entities from your database tables.

The Dali OR Mapping Tool allows you to select the tables from which to generate entities, the table associations, and default entity properties. You can then customize each individual entity being generated.

### 6.3 Conversion of Java Projects to JPA Projects

The Dali OR Mapping Tool now uses the Eclipse [Configure](#) menu to convert existing Java projects to JPA projects.

See "Converting a Java Project to a JPA Project" for more information.

### 6.4 EclipseLink 1.1 Support

Release 2.2 provides support of the following EclipseLink 1.1 features.

- EclipseLink XML mapping file and its configuration options
One-to-Many mapping \textit{without} a join table.

EclipseLink (the Eclipse Persistence Services Project) is a complete persistence framework. Refer to \url{http://www.eclipse.org/eclipselink/} for more information.

6.5 Table-per-Concrete-Class Inheritance

The Dali OR Mapping Tool now supports the JPA 1.0-optional table-per-concrete-class inheritance option. This functionality may be required in JPA 2.0.
Copyright © 2006, 2009, Oracle. All rights reserved.

This program and the accompanying materials are made available under the terms of the Eclipse Public License v1.0 which accompanies this distribution, and is available at:


7.1 About this content

Terms and conditions regarding the use of this guide.

December, 2008

License
The Eclipse Foundation makes available all content in this plug-in ("Content"). Unless otherwise indicated below, the Content is provided to you under the terms and conditions of the Eclipse Public License Version 1.0 ("EPL"). A copy of the EPL is available at http://www.eclipse.org/legal/epl-v10.html. For purposes of the EPL, "Program" will mean the Content.

If you did not receive this Content directly from the Eclipse Foundation, the Content is being redistributed by another party ("Redistributor") and different terms and conditions may apply to your use of any object code in the Content. Check the Redistributor’s license that was provided with the Content. If no such license exists, contact the Redistributor. Unless otherwise indicated below, the terms and conditions of the EPL still apply to any source code in the Content and such source code may be obtained at http://www.eclipse.org.
Annotations

@Basic, 3-22
@Column, 4-12
@DiscriminatorColumn, 3-20, 4-11
@DiscriminatorValue, 3-20, 4-11
@Embeddable, 3-16
@Embedded, 3-23
@EmbeddedId, 3-24
@Entity, 3-15
@Enumerated, 4-13
@GeneratedValue, 4-15
@Id, 3-24
@Inheritance, 3-19
@JoinColumn, 3-28, 3-30, 4-14, 4-15
@Lob, 4-13
@ManyToMany, 3-26
@ManyToOne, 3-27
@MappedSuperclass, 3-17
@NamedQuery, 3-21
@OneToMany, 3-28
@OneToOne, 3-29
@OrderBy, 4-14
@SequenceGenerator, 4-16
@Temporal, 4-13
@Transient, 3-30
@Version, 3-30

A

annotations. See specific annotation.
architecture of Dali feature, 4-33
association tables, 4-8
attribute overrides, 4-10
Attribute Overrides, in Java Details view, 4-10
attributes
  JPA Details view, 4-11
  mapping, 2-1

B

basic mapping
  @Basic, 3-22
  about, 3-22

See also mappings

C

caching, 4-22
cardinality, association tables, 4-9
classes
  adding persistence to, 3-15
  embeddable, 3-16
  entity, 3-15
  managed, 3-11
  managing persistent classes, 4-3
  mapped superclass, 3-17
  synchronizing, 3-12
columns
  discriminator, 3-20, 4-11
  join, 3-28, 3-30, 4-14, 4-15
  mapping to, 4-12
  value, 3-20, 4-11
  connection pool, 3-11
  converting Java project to JPA, 3-5
  Create a JPA Project Wizard, 3-2
  Create New JPA Project wizard, 4-1

database tables
  generating entities from, 3-31
  database, persistence
    connection, 4-30
    schema, 4-30
developer documentation, Dali, 4-33

E

eager fetch, 4-13
EclipseLink XML mapping file, 3-11
EJB. see persistent entities
  embeddable class
    @Embeddable, 3-16
    about, 3-16
  embedded ID mapping
    @EmbeddedId, 3-24
    about, 3-24
  embedded mapping
    @Embedded, 3-23
    about, 3-23
  entities
@Entity annotation, 3-15
about, 2-1
creating, 3-7
customizable, 4-8
embeddable, 3-16
from tables, 3-31, 4-6
generating, 4-7
JPA Details view, 4-9
mapped superclass, 3-17
mapping, 1-3
persistence, 1-3
persistent, 3-15
secondary tables, 4-10
Entity Class page, 3-8
Entity Properties page, 3-9
@Enumerated, 4-13
enumerated, 4-13
error messages, Dali, 3-36
extension points, Dali feature, 4-33

F
fetch type, 4-13

G
Generate Entities from Tables dialog, 3-31, 4-6
generated values
  ID mappings, 4-15
  sequence, 4-16

H
hints, query, 3-21

I
ID mapping
  @Id, 3-24
  about, 3-24
inheritance
  entity, 3-19, 4-4, 4-10
  joined tables, 3-21
  single table, 3-20
Inheritance, in Java Details view, 4-10
installation, Dali, 1-1

J
Java project, converting to JPA, 3-5
join columns, 4-8, 4-14
joined tables, inheritance, 3-21
JPA Details view
  attributes, 4-11
  entities, 4-9
JPA Development perspective, 4-32
JPA Facet page, 3-4
JPA project
  converting from Java, 3-5
  creating new, 3-1
  implementation, 4-2

page, 3-2
platform, 4-30
JPA Structure view, 4-18

L
lazy fetch, 4-13
library, JPA, 4-2

M
many-to-many mapping
  @ManyToMany, 3-26
  about, 3-26
many-to-one mapping
  @ManyToOne, 3-27
  about, 3-27
mapped superclass
  @MappedSuperclass, 3-17
  about, 3-17
mapping entities, 1-3
mapping file, 3-13, 4-4
mappings
  about, 2-1
  basic, 3-22
  embedded, 3-23
  embedded ID, 3-24
  ID, 3-24
  many-to-many, 3-26
  many-to-one, 3-27
  one-to-many, 3-28
  one-to-one, 3-29
  problems, 3-36
  transient, 3-30
  version, 3-30

N
named queries
  entity, 3-21
  hints, 3-21
  native queries, 3-21
  nonpersistent classes, 3-15
  fields. See transient

O
one-to-many mapping
  @OneToMany, 3-28
  about, 3-28
one-to-one mapping
  @OneToOne, 3-29
  about, 3-29
OR (object-relational) mappings. See mappings
ordering, 4-14
orm.xml file
  about, 2-2
  creating, 4-3
  managing, 3-12
  sample, 3-12
Index-3

P

persistence
  about, 2-1
  database connection, 4-30
  database schema, 4-30
  entity class, 3-15
  options, 4-30
  provider, 3-11
persistence.xml file
  about, 2-2
  managing, 3-10, 3-14
  sample, 3-10
  synchronizing with classes, 3-12
  persistent entity, 3-15
  perspective, JPA Development, 4-32
  platform, JPA, 4-30
  problems, 3-36
  projects, JPA
    creating new, 1-2, 3-1
    options, 4-30

Q

query hints, 3-21
quick start, Dali, 1-2

R

requirements
  Dali Java Persistence Tools, 1-1
  persistent entities, 3-15

S

schema, database, 4-30
secondary tables, 4-10
Secondary Tables, in Java Details view, 4-10
single table inheritance, 3-20
superclass, 3-17
synchronizing classes with persistence.xml file, 3-12

T

tables
  associations, 4-7, 4-8
  creating entities from, 3-31, 4-6
  inheritance, 3-20
  secondary, 4-10
  @Temporal, 4-13
  temporal, 4-13
  transient mapping
    @Transient, 3-30
    about, 3-30

V

version mapping

@Version, 3-30
about, 3-30
views
  JPA Details view, 4-9, 4-11
  JPA Structure view, 4-18

W

warning messages, Dali, 3-36
Web Tools Platform (WTP), 1-1

X

XML editor, 3-14