Dali Java Persistence Tools

User Guide

Release 2.1 for Eclipse

December 2008

Dali Java Persistence Tools User Guide

Contents

1 Getting started

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 3.4 (http://www.eclipse.org/downloads)
- Java Runtime Environment (JRE) 1.5 (http://java.com)
- Eclipse Web Tools Platform (WTP) 3.0 (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 2.0. 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:

- Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.
- This documentation may contain links to Web sites of other companies or organizations that we do not control. We neither evaluate nor make any representations regarding the accessibility of these Web sites.

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

Related reference

Tips and tricks
What's new

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 JPA Facet page appears.

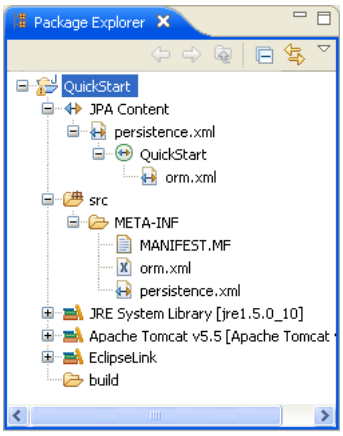
   Note: The Target Runtime is not required for Java SE development.

5. 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.

Figure 1–1 Project in Package Explorer



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 Package 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:

```
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:

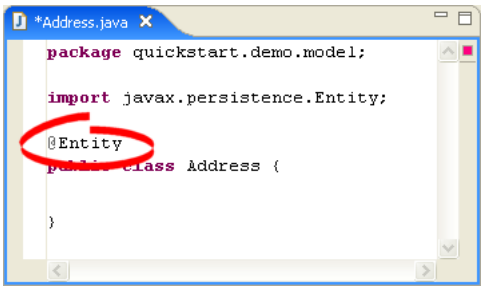
```
NUMBER(10,0) ADDRESS_ID (primary key)
VARCHAR2(80) PROVINCE
VARCHAR2(80) COUNTRY
VARCHAR2(20) P_CODE
VARCHAR2(80) STREET
VARCHAR2(80) CITY
```

---

5. Click **Finish**. With the Create JPA Entity completed, Eclipse displays the **Address** entity in the JPA Structure view.

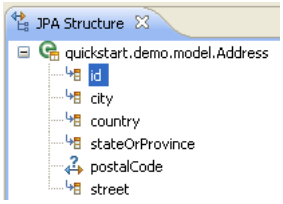
   Address.java includes the `@Entity` annotation, the persistence fields, as well as `getter` and `setter` methods for each of the fields.

Figure 1–2 Address Entity in Address.java



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

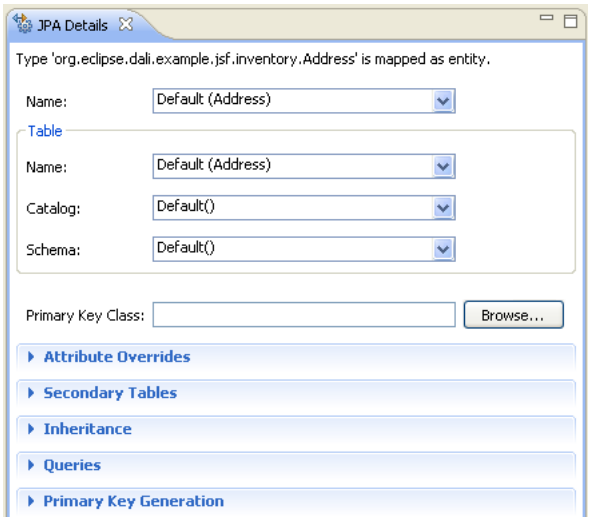
Figure 1–3 Address Entity in the JPA Structure View



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



Tip: After associating the entity with the database table, you should update the `persistence.xml` file to include this JPA entity.

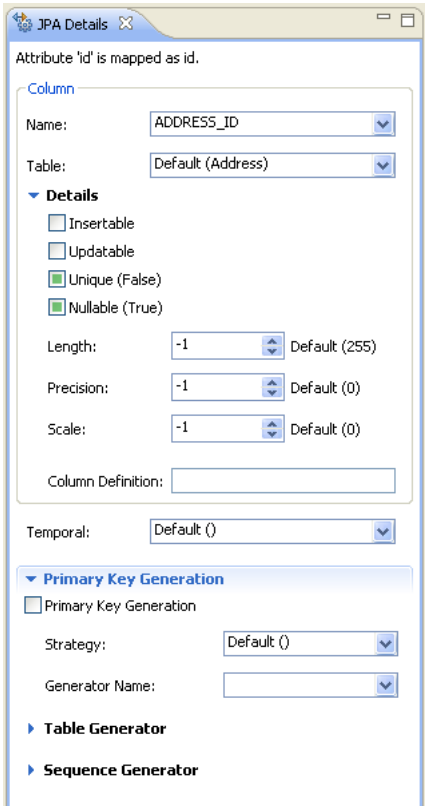
Right-click the `persistence.xml` file in the Package Explorer and select **JPA Tools > Synchronize Classes**. Dali adds the following to the `persistence.xml` file:

```
<class>quickstart.demo.model.Address</class>
```

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:

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

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

Field	Map As	Database Column
city	Basic	CITY
country	Basic	COUNTRY

Field	Map As	Database Column
postalCode	Basic	P_CODE
provinceOrState	Basic	PROVINCE
street	Basic	STREET

Dali automatically maps some fields to the correct database column (such as the city field to the City column) if the names are identical.

2

Concepts

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`
- JSR 220 EJB 3.0 specification: `http://www.jcp.org/en/jsr/detail?id=220`

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.

Related tasks

Mapping an entity

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.

Related tasks

Mapping an entity

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 3-7 for more information.

> **Tip:** To work with multiple persistence units, comment out all but one persistence unit in `persistence.xml`.

Related tasks

Managing the persistence.xml file
Creating a new JPA project

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 3-11 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`.

Related tasks

Managing the orm.xml file
Creating a new JPA project

3

Tasks

This section includes detailed step-by-step procedures for accessing the Dali OR mapping tool functionality.

- Creating a new 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

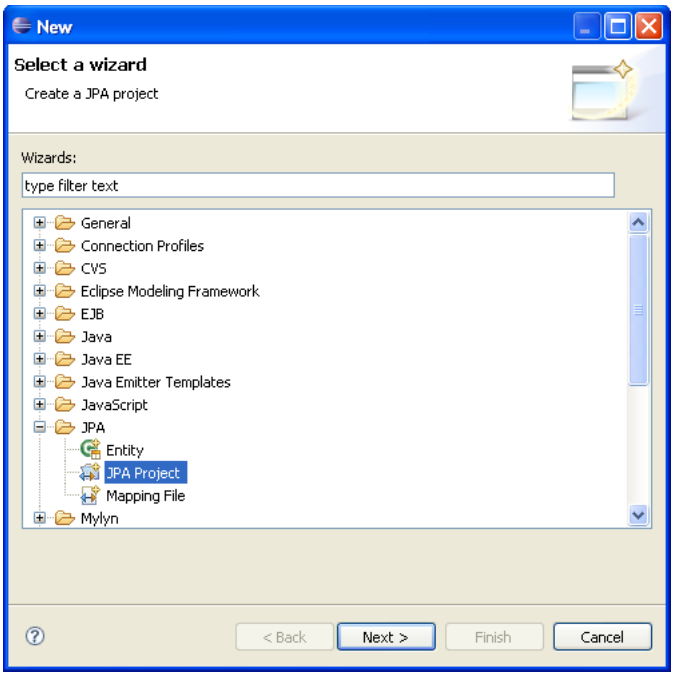
3.1 Creating a new JPA project

Use this procedure to create a new JPA project.

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

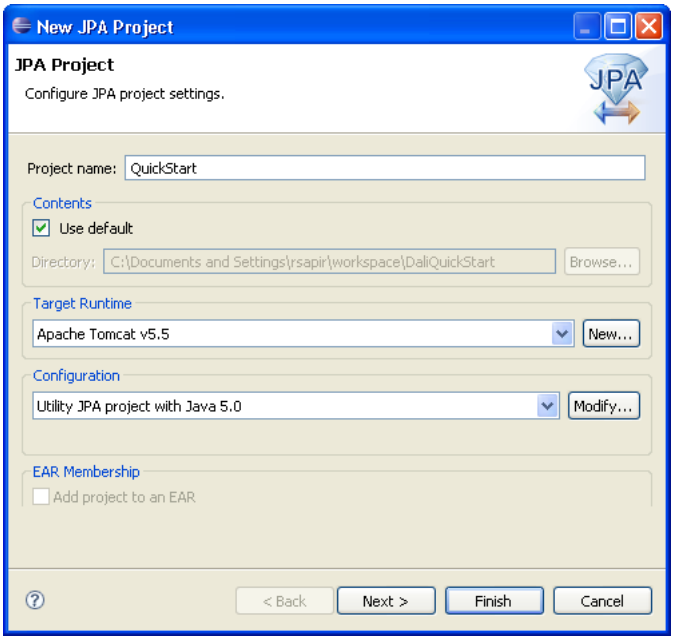
   Tip: You can also select the JPA perspective and then select **File > New > JPA Project**.

Figure 3–1 Selecting the Create a JPA Project wizard



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

Figure 3–2 The JPA Project Page

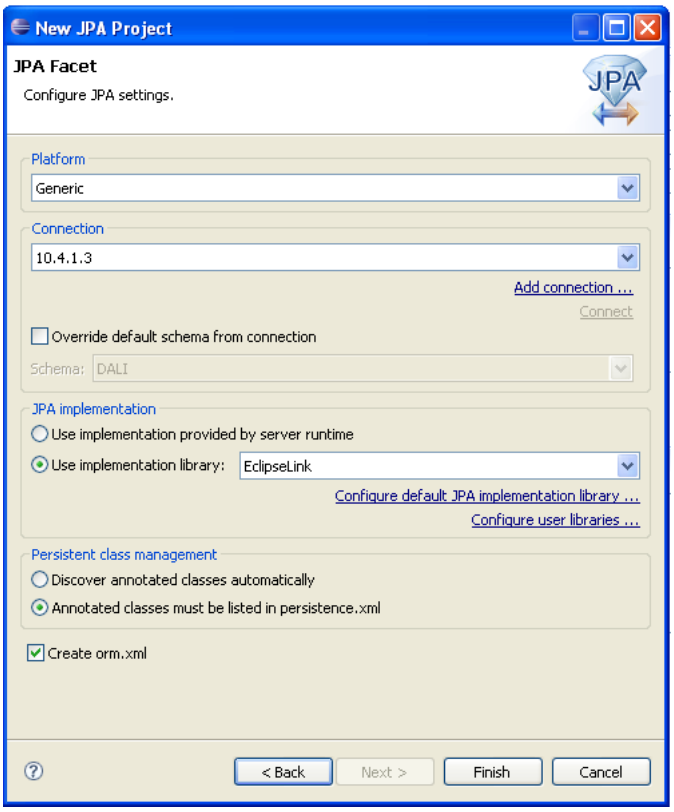


3. Complete the fields on the New JPA Project page to specify the project name and location, target runtime, and pre-defined configuration.

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

4. Click **Next**. JPA Facet page appears.

Figure 3–3 The JPA Facet Page



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

   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.

   Note: If the server runtime does not provide a JPA implementation, you must explicitly select a JPA implementation library.

   To insure the portability of your application, you must explicitly list the managed persistence classes that are included in the persistence unit. If the server supports EJB 3.0, the persistent classes will be discovered automatically.

   Depending on your JPA implementation (for example, Generic or EclipseLink), different options may be available when creating JPA projects.

   ---

6. Click **Finish**. You should now open the JPA Development perspective.

Related reference

Create New JPA Project wizard
JPA Development perspective

Mapping File Wizard

Related tasks

Managing the persistence.xml file
Adding persistence to a class

Related concepts

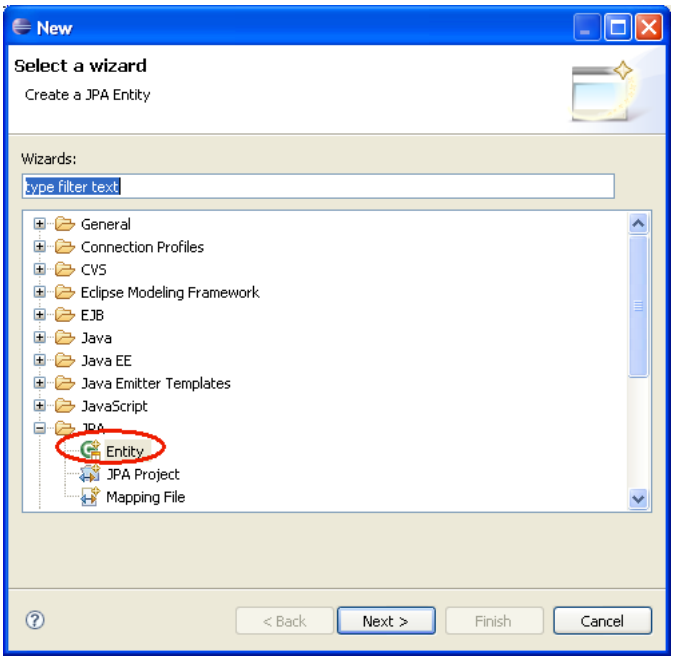
Understanding Java persistence
The persistence.xml file

3.2 Creating a JPA Entity

Use this procedure to create a JPA entity:

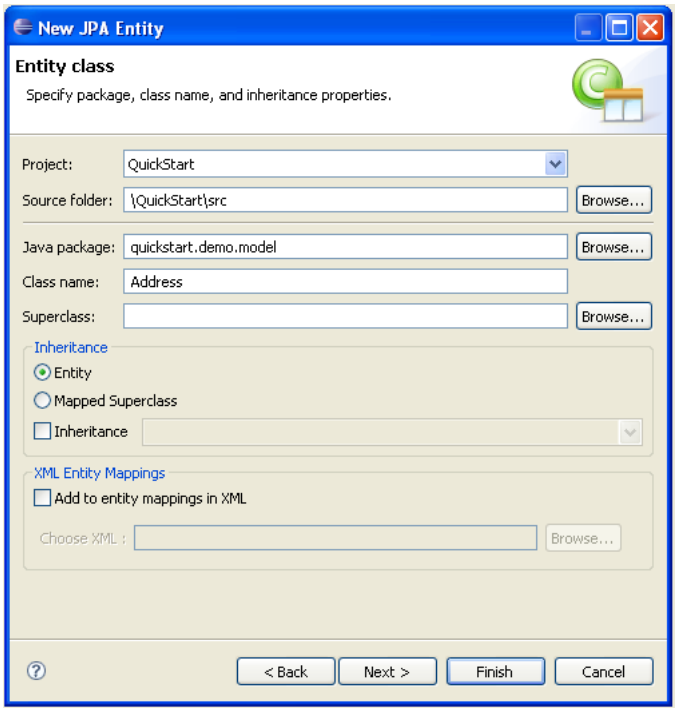
1. From the Navigator or Package Explorer, select the JPA project and then **File > New > Other**. The Select a Wizard dialog appears.

Figure 3–4 Selecting the Create a JPA Entity Wizard



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

Figure 3–5 The Entity Class Page

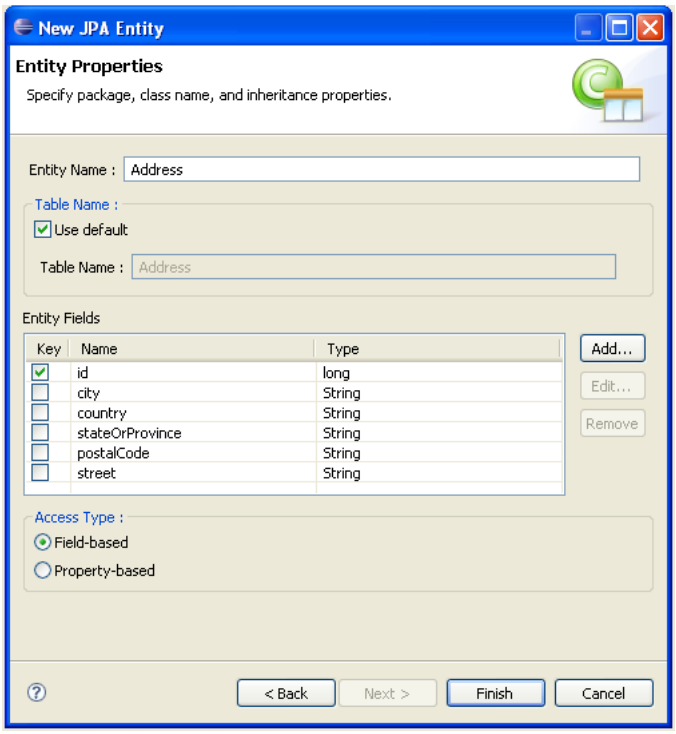


Complete this page as follows:

- 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 3–6 The Entity Properties Page



Alternatively, click **Finish** to complete the entity.

4. Complete the page as follows:

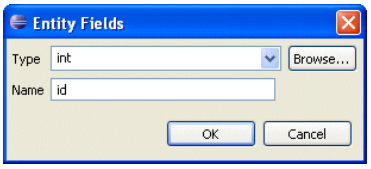
    1. If needed, enter a new name for the entity. Doing so results in adding a `name` attribute to the `@Entity` notation (`@Entity(name="EntityName")`).

    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 3–7 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.

6. If needed, select **Key** to designate the field as a primary key.
7. Select either the **Field-based** access type (the default) or **Property-based** access type.

Related reference

Create JPA Entity wizard
Create New JPA Project wizard
JPA Development perspective

Related tasks

Managing the persistence.xml file
Adding persistence to a class

Related concepts

Understanding Java persistence
The persistence.xml file

3.3 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 `peristence.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.

Property	Description	Element Defined
Name	Enter the name of the persistence unit.	`<persistence-unit name = "<Name>">`
Persistence Provider	Enter the name of the persistence provider.	`<provider>`
Description	Enter a description for this persistence unit. This is an optional property.	`<description>`
Managed Classes	Add or remove the classes managed through the persistence unit.	`<class>`
Exclude Unlisted Classes	Select to include all annotated entity classes in the root of the persistence unit.	`<exclude-unlisted-classes>`
XML Mapping Files	Add or remove the object/relational mapping XML files define the classes managed through the persistence unit.	`<mapping-file>`

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.

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.

Related reference

persistence.xml Editor

Related tasks

Using the XML Editor to edit the persistence.xml file

Related concepts

The persistence.xml file

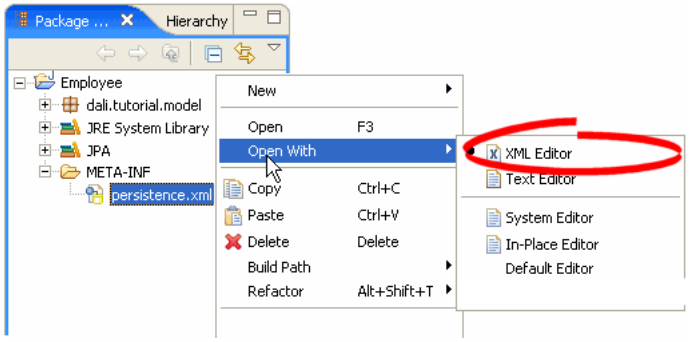
3.3.1 Using the XML Editor to edit the persistence.xml file

You can work with the `persistence.xml` by using the XML Editor.

Use this procedure to work with the `persistence.xml` file:

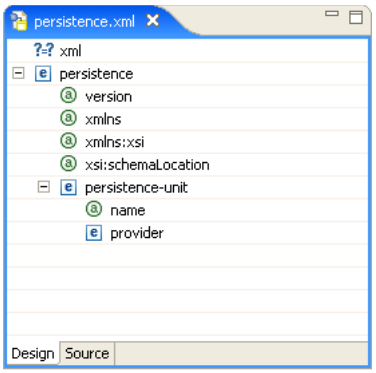
1. Right-click the `persistence.xml` file in the Package Explorer and select **Open With > XML Editor**.

Figure 3–8 Opening the XML Editor



2. Use the XML Editor to edit the `persistence.xml` file.

Figure 3–9 XML Editor



Related tasks

Using the XML Editor to edit the persistence.xml file
Working with XML Files

Related concepts

The persistence.xml file

3.3.2 Synchronizing classes

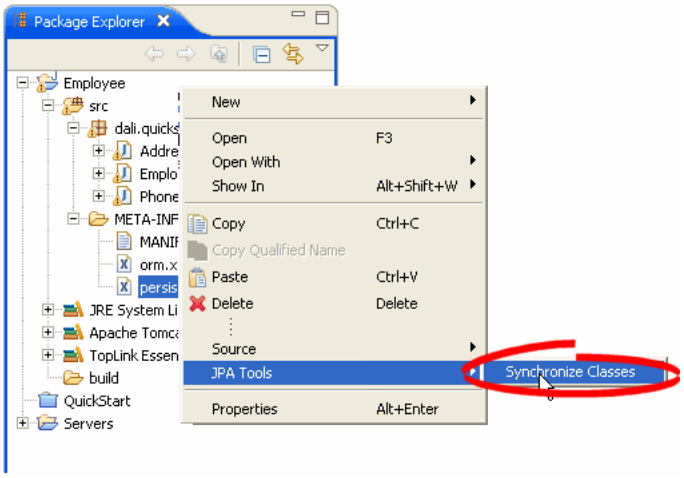
As you work with the classes in your Java project, you will need to update the `persistence.xml` file to reflect the changes.

Use this procedure to synchronize the `persistence.xml` file:

1. Right-click the `persistence.xml` file in the Package Explorer and select **JPA Tools > 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 3–10 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.

Related tasks

Using the XML Editor to edit the persistence.xml file

Related concepts

The persistence.xml file

3.4 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 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>
```

Related reference

Create New JPA Project wizard

Related tasks

Working with orm.xml file

Related concepts

The orm.xml file

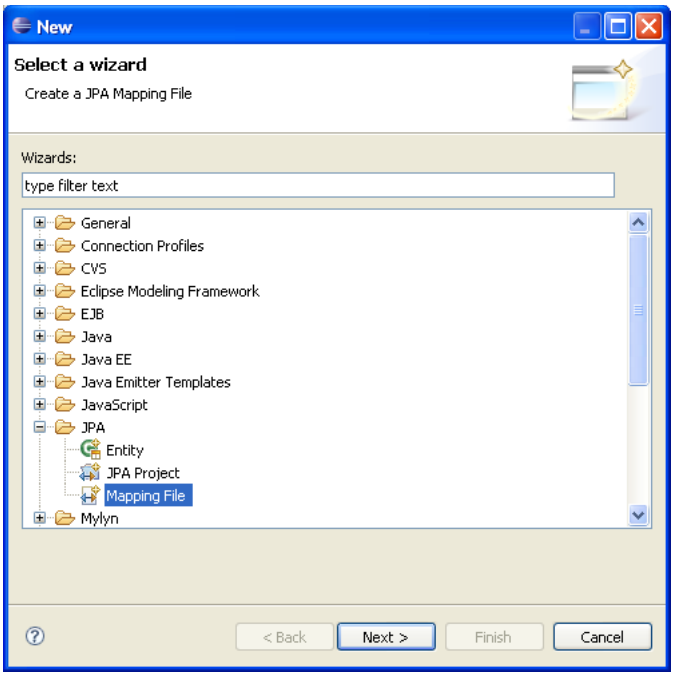
3.4.1 Creating an orm.xml file

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 Package Explorer, select **File > New > Other**. The Select a Wizard dialog appears.

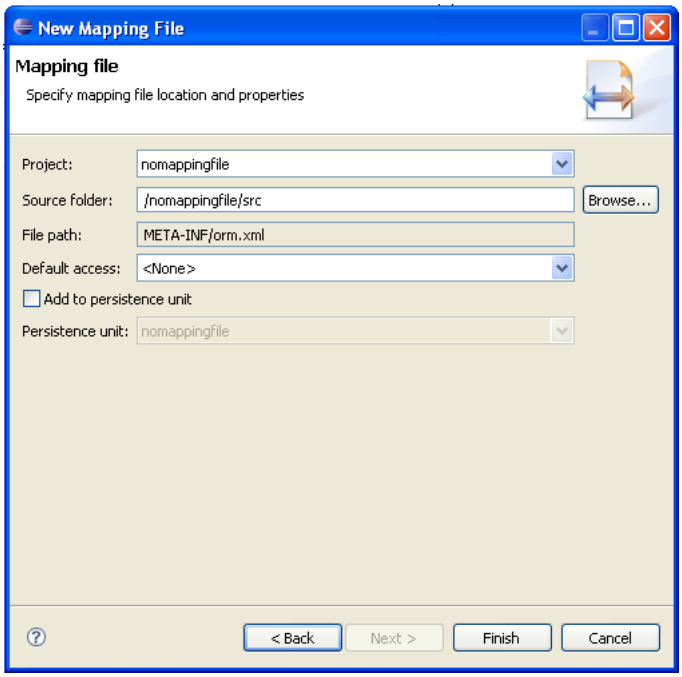
Figure 3–11 The Select a Wizard Dialog



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

   If you are using EclipseLink, you can select **EclipseLink Mapping File**.

Figure 3–12 The Mapping File Page



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).

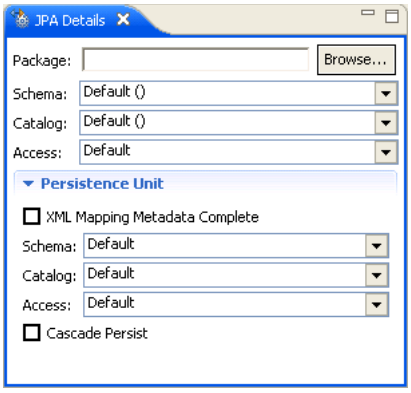
3.4.2 Working with orm.xml file

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 Package 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.

Figure 3–13 JPA Details view for EntityMappings (orm.xml)



Related tasks

Working with orm.xml file
Working with XML Files

Related concepts

The orm.xml file

3.5 Adding persistence to a class

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

- Entity
- Embeddable
- Mapped superclass

Related tasks

Specifying additional tables
Specifying entity inheritance
Mapping an entity

Related concepts

Understanding Java persistence
The orm.xml file
The persistence.xml file

3.5.1 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*:

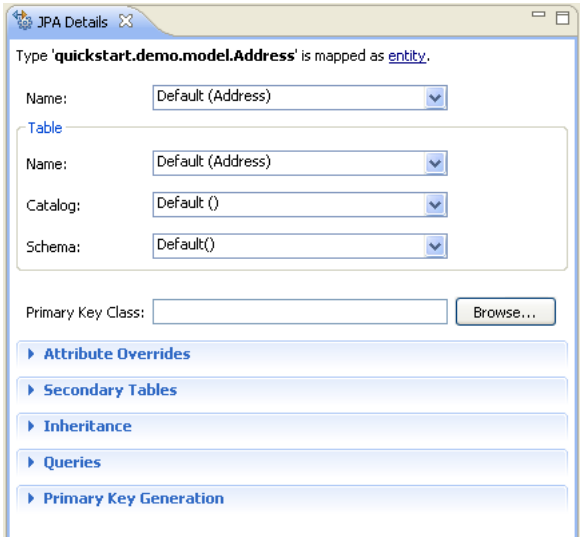
- A no-arg constructor (public or protected); the entity class may have other constructors as well.

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 Package 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.

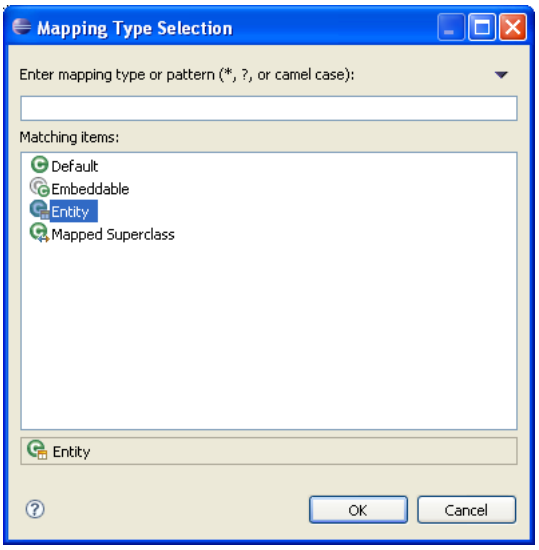
Figure 3–14 The Mapping Type Hyperlink



> **Tip:** You can also change (or add) persistence for an entity by right-clicking the class in the JPA Structure View and then clicking **Map As > Entity**.

4. Select **Entity** from the Mapping Type Selection dialog and then click **OK**.

Figure 3–15 The Mapping Type Selection Dialog



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

Related tasks

Adding persistence to a class
Specifying additional tables
Specifying entity inheritance

3.5.2 Embeddable

An **Embedded** 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.

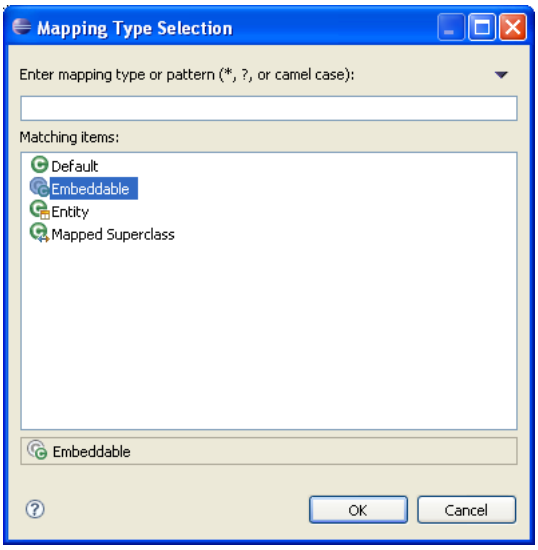
To override the mapping information for a specific subclass, use the `@AttributeOverride` annotation for that specific class.

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 Package 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**.

Figure 3–16 Mapping Type Selection Dialog (Embeddable)



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

Related tasks

Adding persistence to a class
Specifying additional tables
Specifying entity inheritance

3.5.3 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

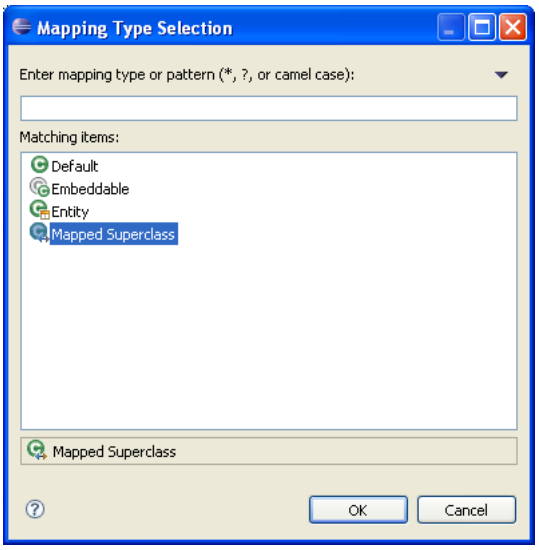
A mapped superclass does not have a defined database table. Instead, its mapping information is derived from its superclass. To override the mapping information for a specific subclass, use the `@AttributeOverride` annotation for that specific class.

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 Package 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**.

Figure 3–17 Mapping Type Selection Dialog (Mapped Superclass)



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

Related tasks

Adding persistence to a class
Specifying additional tables
Specifying entity inheritance

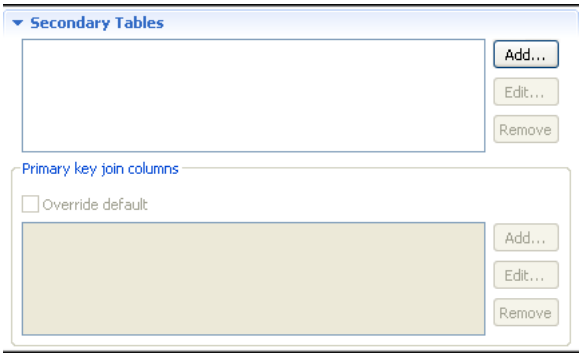
3.6 Specifying additional tables

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 Package Explorer.
2. In the JPA Details view, select the **Secondary Tables** information.

Figure 3–18 Specifying Secondary Tables



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:

```
@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:

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

Related tasks

Adding persistence to a class

Related concepts

Understanding Java persistence

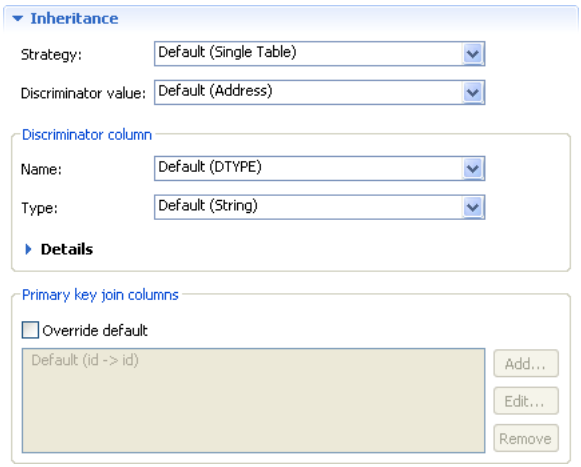
3.7 Specifying entity inheritance

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

Figure 3–19 Specifying Inheritance



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 4-7 for additional details.

Property	Description	Default
Discriminator Column	Name of the discriminator column when using a **Single** or **Joined** inheritance strategy. This field corresponds to the `@DiscriminatorColumn` annotation.	
Discriminator Type	Set the discriminator type to `Char` or `Integer` (instead of its default: `String`). The **Discriminator Value** must conform to this type.	String
Discriminator Value	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.	
Override Default	Use this field to specify custom primary key join columns. This field corresponds to the `@PrimaryKeyJoinClumn` annotation.	

Eclipse adds the following annotations the entity field:

```
@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 3–20 Single Table Inheritance

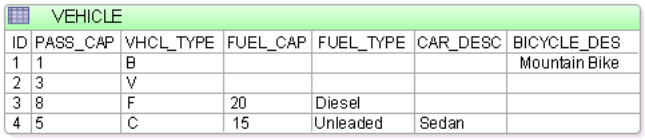
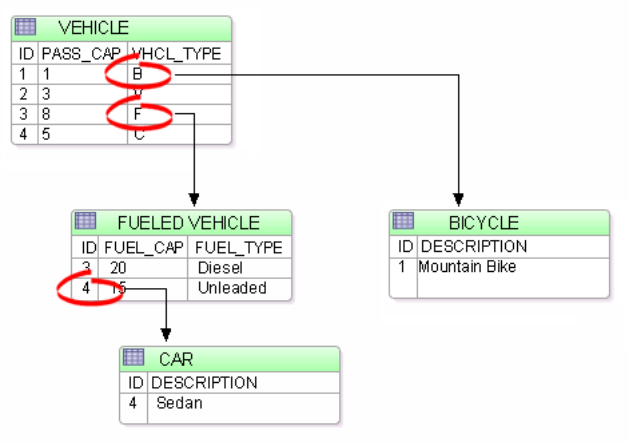


Figure 3–21 Joined Table Inheritance



Related tasks

Adding persistence to a class

Related concepts

Understanding Java persistence

3.8 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.

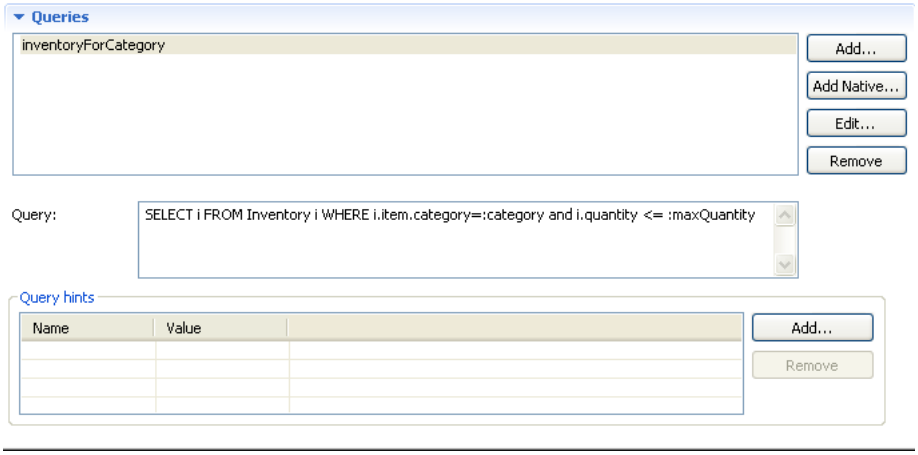
Use this procedure to add `@NamedQuery` and `@NamedNativeQuery` annotations to the entity.

To create a named query:

1. Select the entity in the Package 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**.

Figure 3–22 Entering a Named Query



3.9 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

Related concepts

Understanding OR mappings

3.9.1 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.

Property	Description	Default
Entity Map Hyperlink	Defines this mapping as a **Basic Mapping**. This corresponds to the `@Basic` annotation.	Basic
Column	The database column mapped to the entity attribute. See "Column" on page 4-9 for details.	By default, the Column is assumed to be named identically to the attribute and always included in the `INSERT` and `UPDATE` statements.
Table	Name of the database table.	
Fetch	Defines how data is loaded from the database. See "Fetch Type" on page 4-9 for details. ■ Eager ■ Lazy	Eager
Optional	Specifies if this field is can be null.	Yes
Temporal	Specifies the type of data. See "Temporal" on page 4-10 for details. ■ Date ■ Time ■ Timestamp	
Lob	Specifies if this is a large objects (BLOB or CLOB). See "Lob" on page 4-10 for details.	

Eclipse adds the following annotations to the field:

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

Related tasks

Mapping an entity

Related reference

JPA Structure view
JPA Details view (for attributes)

Related concepts

Understanding OR mappings
Understanding EJB 3.0 Java Persistence API

3.9.2 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.

Property	Description	Default
Entity Mapping Hyperlink	Defines this mapping as a **Embedded**. This corresponds to the `@Embedded` annotation.	Embedded
Attribute Overrides	Specify to override the default mapping of an entity's attribute. Select **Override Default**.	
Columns	The database column (and its table) mapped to the entity attribute. See "Column" on page 4-9 for details. ■ Name – Name of the database column. ■ Table – Name of the database table.	

Eclipse adds the following annotations to the field:

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

Related tasks

Mapping an entity

Related reference

JPA Structure view
JPA Details view (for attributes)

Related concepts

Understanding OR mappings
Understanding EJB 3.0 Java Persistence API

3.9.3 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.

Property	Description	Default
Entity Mapping Hyperlink	Defines this mapping as a **Embedded Id**. This corresponds to the `@EmbeddedId` annotation.	Embedded Id

Eclipse adds the following annotations to the field:

```
@EmbeddedId
```

Related tasks

Mapping an entity

Related reference

JPA Structure view
JPA Details view (for attributes)

Related concepts

Understanding OR mappings
Understanding EJB 3.0 Java Persistence API

3.9.4 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 filed 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.

Property	Description	Default
Entity Mapping Hyperlink	Defines this mapping as an **ID Mapping**. This field corresponds to the `@Id` annotation.	ID
Column	The database column mapped to the entity attribute. See "Column" on page 4-9 for details.	By default, the Column is assumed to be named identically to the attribute.
Table	The database table mapped to the entity attribute.	By default, the Table is assumed to be identical to the table associated with the entity.

Property	Description	Default
Temporal	Specifies the type of data. See "Temporal" on page 4-10 for details. ■ Date ■ Time ■ Timestamp	

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

Property	Description	Default
Primary Key Generation	These fields define how the primary key is generated.	
Strategy	See "Primary Key Generation" on page 4-12 for details. ■ Auto ■ Sequence ■ Identity ■ Table	Auto
Generator Name	Name of the primary key generator specified in the **Strategy**	

Additional fields will appear in the Primary Key Generation information area, depending on the selected Strategy. See "JPA Details view (for attributes)" on page 4-8 for additional information.

Eclipse adds the following annotations to the field:

```
@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>",
    sequenceName="<SEQUENCE_GENERATOR_SEQUENCE>")
```

Related tasks

Mapping an entity

Related reference

JPA Structure view
JPA Details view (for attributes)

Related concepts

Understanding OR mappings
Understanding EJB 3.0 Java Persistence API

3.9.5 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.

Property	Description	Default
Mapping Entity Hyperlink	Defines this mapping as a **Many to Many Mapping**. This field corresponds to the `@ManyToMany` annotation.	Many to Many
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	Defines how data is loaded from the database. See "Fetch Type" on page 4-9 for details. ■ Eager ■ Lazy	Lazy
Mapped By	The database field that owns the relationship.	
Order By	Specify the default order for objects returned from a query. See "Order By" on page 4-10 for details. ■ No ordering ■ Primary key ■ Custom	No ordering

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

Property	Description	Default
Name	Name of the join table that contains the foreign key column.	You must specify the join table on the owning side. By default, the name is assumed to be the primary tables associated with the entities concatenated with an underscore.
Join Columns	Select **Override Default**, then Add, Edit, or Remove the join columns.	By default, the name is assumed to be the primary tables associated with the entities concatenated with an underscore.

Property	Description	Default
Inverse Join Columns	Select **Override Default**, then Add, Edit, or Remove the join columns.	By default, the mapping is assumed to have a single join.

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:

```
@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>")
```

Related tasks

Mapping an entity

Related reference

JPA Structure view
JPA Details view (for attributes)

Related concepts

Understanding OR mappings
Understanding EJB 3.0 Java Persistence API

3.9.6 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.

Property	Description	Default
Mapping Entity Hyperlink	Defines mapping as **Many-to-One**. This corresponds to the `@ManyToOne` annotation.	Many-to-One
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	Defines how data is loaded from the database. See "Fetch Type" on page 4-9 for details. ▪ Eager ▪ Lazy	Eager

Property	Description	Default
Cascade	See "Cascade Type" on page 4-10 for details. ■ Default ■ All ■ Persist ■ Merge ■ Remove	Default
Optional	Specifies if this field is can be null.	Yes

4. Use this table to complete the fields on the Join Columns Information tab in the JPA Details view.

Property	Description	Default
Join Column	Specify a mapped column for joining an entity association. This field corresponds to the `@JoinColum` attribute. Select **Override Default**, then Add, Edit, or Remove the join columns.	By default, the mapping is assumed to have a single join.

Eclipse adds the following annotations to the field:

```
@JoinTable(joinColumns=@JoinColumn(name="<JOIN_COLUMN>"),
    name = "<JOIN_TABLE_NAME>")
@ManyToOne(targetEntity=<TARGET_ENTITY>, fetch=<FETCH_TYPE>,
    cascade=<CASCADE_TYPE>)
```

Related tasks

Mapping an entity

Related reference

JPA Structure view
JPA Details view (for attributes)

Related concepts

Understanding OR mappings
Understanding EJB 3.0 Java Persistence API

3.9.7 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.

Property	Description	Default
Mapping Entity Type Hyperlink	Defines mapping as **One-to-Many**. This corresponds to the `@OneToMany` annotation.	One-to-Many
Target Entity	The entity to which this attribute is mapped.	
Cascade	See "Cascade Type" on page 4-10 for details. ■ Default ■ All ■ Persist ■ Merge ■ Remove	
Fetch	Defines how data is loaded from the database. See "Fetch Type" on page 4-9 for details. ■ Eager ■ Lazy	Eager
Mapped By	The database field that owns the relationship.	
Order By	Specify the default order for objects returned from a query. See "Order By" on page 4-10 for details. ■ No ordering ■ Primary key ■ Custom	No ordering

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

Property	Description	Default
Name	Name of the join table	By default, the name is assumed to be the primary tables associated with the entities concatenated with an underscore.
Join Columns	Specify two or more join columns (that is, a primary key).	
Inverse Join Columns	The join column on the owned (or inverse) side of the association: the owned entity's primary key column.	

Eclipse adds the following annotations to the field:

```
@OneToMany(targetEntity=<TARGET_ENTITY>)
@Column(name="<COLUMN>")
```

```
@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>"))
```

Related tasks

Mapping an entity

Related reference

JPA Structure view
JPA Details view (for attributes)

Related concepts

Understanding OR mappings
Understanding EJB 3.0 Java Persistence API

3.9.8 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.

Property	Description	Default
Mapped Entity Hyperlink	Defines mapping as **One-to-One**. This corresponds to the `@OneToOne` annotation.	One-to-One
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 4-9 for details. ▪ Eager ▪ Lazy	Eager
Mapped By	The database field that owns the relationship.	

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

Property	Description	Default
Join Column	Specify a mapped column for joining an entity association. This field corresponds to the `@JoinColum` attribute. Select **Override Default**, then Add, Edit, or Remove the join columns.	By default, the mapping is assumed to have a single join.

Eclipse adds the following annotations to the field:

```
@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>)
```

Related tasks

Mapping an entity

Related reference

JPA Structure view
JPA Details view (for attributes)

Related concepts

Understanding OR mappings
Understanding EJB 3.0 Java Persistence API

3.9.9 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:

```
@Transient
```

Related tasks

Mapping an entity

Related reference

JPA Structure view
JPA Details view (for attributes)

Related concepts

Understanding OR mappings
Understanding EJB 3.0 Java Persistence API

3.9.10 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.

Property	Description	Default
Mapped Entity Hyperlink	Defines the mapping as Version. This corresponds to the `@Version` annotation.	Version
Column	The database column mapped to the entity attribute. See "Column" on page 4-9 for details.	By default, the Column is assumed to be named identically to the attribute and always included in the `INSERT` and `UPDATE` statements.
Table	Name of the database table. This must be the primary table associated with the attribute's entity.	
Temporal	Specifies the type of data. See "Temporal" on page 4-10 for details. ■ Date ■ Time ■ Timestamp	

Eclipse adds the following annotations to the field:

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

Related tasks

Mapping an entity

Related reference

JPA Structure view
JPA Details view (for attributes)

Related concepts

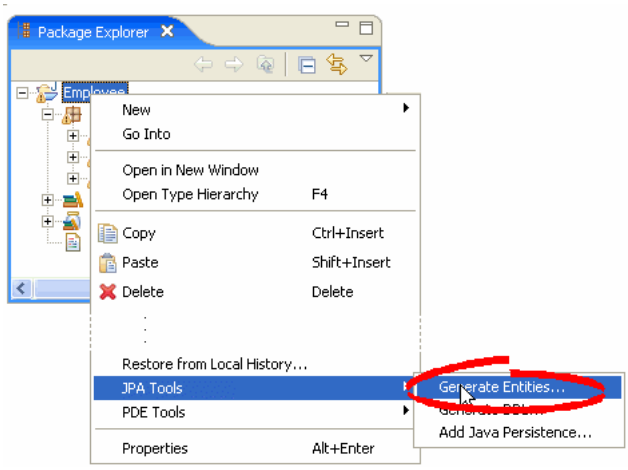
Understanding OR mappings
Understanding EJB 3.0 Java Persistence API

3.10 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 Package Explorer and select **JPA Tools > Generate Entities**.

Figure 3–23 Generating Entities



2. If you are not currently connected to the database, the Database Connection page appears. Select your database connection and schema, and click **Reconnect**.

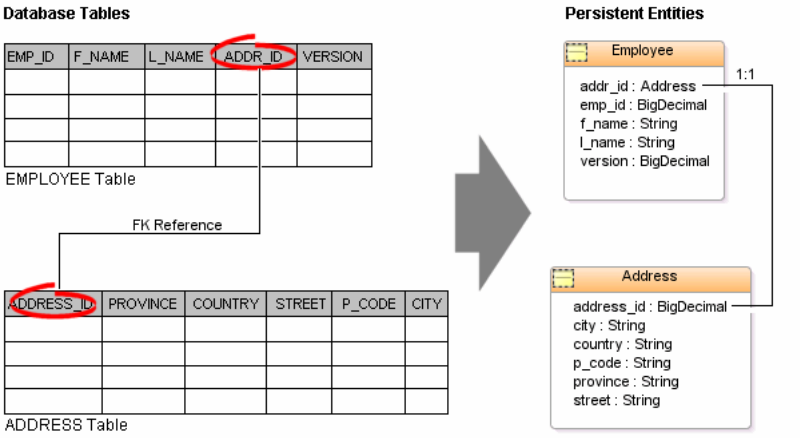
   To create a new database connection, click **Add connection**.

   After connecting to the database, click **Next**.

3. On the Generate Entities from Tables dialog dialog, select the tables from which to generate Java persistent entities and 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 3–24 illustrates how Eclipse generates entities from tables.

Figure 3–24 Generating Entities from Tables



Related tasks

Creating a new JPA project

Related reference

Project Properties page – JPA Options

3.11 Generating DDL from Entities

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

To generate a DDL script:

Right-click the JPA project in the Package Explorer and select **JPA Tools > Generate DDL**.

Related tasks

Creating a JPA Entity

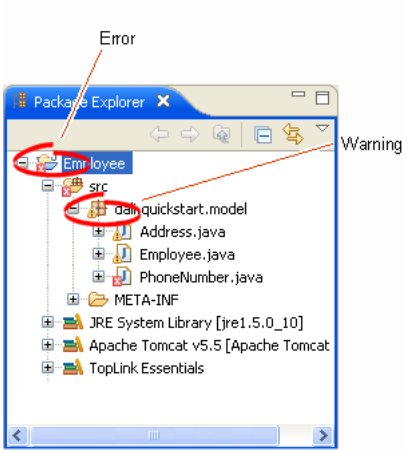
Related reference

Project Properties page – JPA Options

3.12 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 3–25 Sample Errors and Warnings



This section contains information on the following:

- Error messages
- Warning messages

Related concepts

Problems view

3.12.1 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 3-21 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 3-13.

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 3-21.

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 3-13 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 3-7 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 3-11 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 3-7 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.

See "JPA Details view (for attributes)" on page 4-8 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 3-11 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 3-13.

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.

Related concepts

Problems view

3.12.2 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 3-37 for information on specifying a database connection.

Related concepts

Problems view

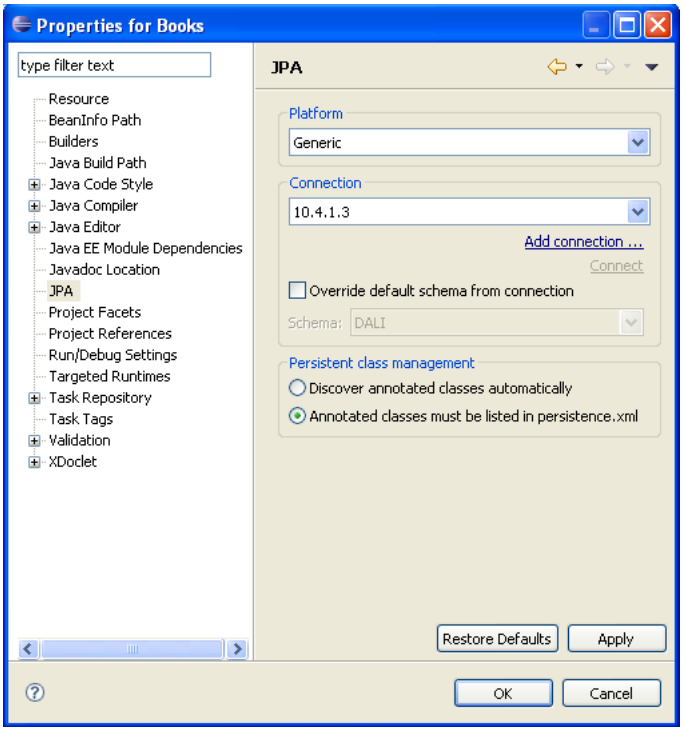
3.13 Modifying persistent project properties

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 vender-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 3–26 The Properties Page



2. Use this table to complete the remaining fields on the Properties – JPA page and click **OK**.

Property	Description
Platform	Select the vendor-specific platform for the JPA implementation.
Database Connection	Database connection to use to store the persistent entities. To create a new connection, click **Add Connection**.
Override default schema from connection	Select a schema other than the default one derived from the connection information. Use this option if the default schema cannot be used. For example, use this option in cases where the deployment login differs from the design-time login.

To create a new connection, click **Add connections**.

- Related tasks

  Creating a new JPA project

- Related reference

  Project Properties page – JPA Options

- Related concepts

  Understanding Java persistence

4 Reference

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

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
- 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.

Property	Description	Default
Project name	Name of the Eclipse JPA project.	
Project contents	Location of the workspace in which to save the project. Unselect The **Use Default** option and click **Browse** to select a new location.	Current workspace

Property	Description	Default
Target runtime	Select a pre-defined target for the project. Click **New** to create a new environment with the New Server Runtime wizard.	
Configurations	Select a project configuration with pre-defined facets. Select **<custom>** to manually select the facets for this project.	Utility JPA project with Java 5.0
EAR membership	Specify if this project should be included in an EAR file for deployment. Select the **EAR Project Name**, or click **New** to create a new EAR project.	

4.1.1.2 JPA Facet page

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

Property	Description	Default
Platform	Vendor-specific JPA implementation.	EclipseLink
Connection	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 **Add connection** to create a new database connection.	
Override default schema from connection	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.	The value calculated by Dali.
JPA Implementation	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 **Configure default JPA implementation library** to create a default library for the project or click **Configure user libraries** to define additional libraries. Depending on your JPA implementation (for example, Generic or EclipseLink), different options may be available when working with JPA projects.	Determined by server.
Persistent class management	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.	Determined by server.

Property	Description	Default
Create orm.xml	Specify if Dali should create a default `orm.xml` file for your entity mappings and persistence unit defaults.	Selected

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

4.1.2.1 Entity Class page

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

Table 4–1

Property	Description	Default
Project	The name of t he JPA project.	
Source Folder	The location of the JPA project's src folder.	
Java Package	The name of the class package.	
Class name	The name of the Java class.	
Superclass	Select the superclass.	
Inheritance	Because the wizard creates a Java class with an `@Entity` notation, the **Entity** option is selected by default. Select **Mapped Superclass** if you defined a superclass. To add an `@Inheritance` notation to the entity, select **Inheritance** and then select one of the inheritance mapping strategies (described in JSR 220): ■ SINGLE_TABLE -- All classes in a hierarchy as mapped to a single table. This annotation is without an attribute for the inheritance strategy. ■ TABLE_PER_CLASS -- Each class is mapped to a separate table. ■ 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.	Entity

Table 4–1 (Cont.)

Property	Description	Default
XML Entity Mappings	Select **Add to entity mappings in XML** to create XML mappings in `orm.xml`, rather than annotations.	

4.1.2.2 Entity Properties page

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

Table 4–2

Property	Description	Default
Entity Name	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="EntityName").`	Determined by server.
Table Name	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.	Use default.
Entity Fields	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.	
Access Type	Select whether the entity's access to instance variables is field-based or property-based, as defined in the JPA specification. ▪ **Field-based** – Instance variables are accessed directly. All non-transient instance variables are persistent. ▪ **Property-based** – Persistent state accessed through the property accessor methods. The property accessor methods must be **public** or **private**.	Field-based

Related tasks

Creating a JPA Entity
Adding persistence to a class

Related reference

JPA Details view (for entities)

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 4–3 Mapping File Wizard Properties

Property	Description	Default
Project	The name of the JPA project.	Selected.
Source folder	The location of the project's src folder. If needed, click **Browse** to point the wizard to the src file's location.	Selected.
File Path	The location for the new `orm.xml` file.	Selected.
Default Access	Select whether the access to the entity is field-based or property-based, as defined in JPA specification. ■ None – No access type specified. ■ **Property-based** – Persistent state accessed through the property accessor methods. The property accessor methods must be **public** or **private**. ■ **Field-based** – Instance variables are accessed directly. All non-transient instance variables are persistent.	None
Add to persistence unit	Designates the persistence unit for this object map file.	Selected.

4.1.4 Generate DDL from Entities Wizard

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.

4.2 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.

Property	Description	Default	Available for Entity Type
Mapping Type Hyperlink	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.	Entity	Entity, Embeddable, and Mapped superclass
Name	The name of this entity. By default, the class name is used as the entity name.		Entity
Table	The default database table information for this entity. These fields can be overridden by the information in the Attribute overrides area.		Entity
Name	The name of the primary database table associated with the entity.		Entity
Catalog	The database catalog that contains the **Table**.	As defined in `orm.xml`.	Entity
Schema	The database schema that contains the **Table**.	As defined in `orm.xml`.	Entity

Related tasks

Adding persistence to a class

Related reference

JPA Details view (for entities)

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.

Property	Description	Default	Available for Entity Type
Attribute Overrides	Specify a property or field to be overridden (from the default mappings). Select **Override Default**.		Entity
Join Columns			Entity

Related tasks

General information
Adding persistence to a class

Related reference

JPA Details view (for entities)

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 3-18 for additional information.

Related tasks

Specifying additional tables
Adding persistence to a class

Related reference

JPA Details view (for entities)

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.

Property	Description	Default
Strategy	Specify the strategy to use when mapping a class or class hierarchy: ▪ Single table – All classes in the hierarchy are mapped to a single table. ▪ Joined – The root of the hierarchy is mapped to a single table; each child maps to its own table. ▪ Table per class – Each class is mapped to a separate table.	Single table
Discriminator Column	Use to specify the name of the discriminator column when using a **Single** or **Joined** inheritance strategy.	
Discriminator Type	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.	String
Discriminator Value	Specify the discriminator value used to differentiate an entity in this inheritance hierarchy. The value must conform to the specified **Discriminator Type**.	

Property	Description	Default
Primary Key Join Columns	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.	

Refer to "Specifying entity inheritance" on page 3-18 for additional information.

Related tasks

Specifying entity inheritance
Adding persistence to a class

Related reference

JPA Details view (for entities)

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 3-20 for additional information.

Related tasks

Creating Named Queries

Related reference

JPA Details view (for entities)

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

See "Mapping an entity" on page 3-21 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.

Property	Description	Default	Available for Mapping Type
Mapping Type Hyperlink	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.	Basic	All mapping types
Column	The database column that contains the value for the attribute. This field corresponds to the `@Column` annotation.	By default, the Column is assumed to be named identically to the attribute.	Basic mapping, Embedded mapping, ID mapping, Version mapping
Name	Name of the database column. This field corresponds to the `@Column` annotation.		Basic mapping, Embedded mapping, ID mapping
Table	Name of the database table that contains the selected column.		Basic mapping, Embedded mapping, ID mapping
Insertable	Specifies if the column is always included in `SQL INSERT` statements.	True	Basic mapping, Embedded mapping, ID mapping
Updatable	Specifies if this column is always included in `SQL UPDATE` statements.	True	Basic mapping, Embedded mapping, ID mapping
Unique	Sets the `UNIQUE` constraint for the column.	False	Basic mapping, Embedded mapping, ID mapping
Nullable	Specifies if the column allows null values.	True	Basic mapping, Embedded mapping, ID mapping
Length	Sets the column length.	255	Basic mapping, Embedded mapping, ID mapping
Precision	Sets the precision for the column values.	0	Basic mapping, Embedded mapping, ID mapping
Scale	Sets the number of digits that appear to the right of the decimal point.	0	Basic mapping, Embedded mapping, ID mapping
Column Definition	Define the DDL for a column. This is used when a table is being generated.		Basic mapping, Embedded mapping, ID mapping
Fetch Type	Defines how data is loaded from the database: ■ Eager – Data is loaded in before it is actually needed. ■ Lazy – Data is loaded only when required by the transaction.	Eager	Basic mapping, One-to-one mapping, Many-to-many mapping, Many-to-one mapping
Optional	Specifies if this field is can be null.	Yes	Basic mapping, One-to-one mapping, Many-to-one mapping

Property	Description	Default	Available for Mapping Type
Lob	Specify if the field is mapped to `java.sql.Clob` or `java.sql.Blob`. This field corresponds to the `@Lob` annotation.		Basic mapping
Temporal	Specifies if this field is one of the following: ■ Date – `java.sql.Date` ■ Time – `java.sql.Time` ■ Timestamp – `java.sql.Timestamp` This field corresponds to the `@Temporal` annotation.		Basic mapping, ID mapping
Enumerated	Specify how to persist enumerated constraints if the `String` value suits your application requirements or to match an existing database schema. ■ ordinal ■ `String` This field corresponds to the `@Enumerated` annotation.	Ordinal	
Target Entity	The persistent entity to which the attribute is mapped.		One-to-one mapping, One-to-many mapping Many-to-many mapping, Many-to-one mapping
Cascade Type	Specify which operations are propagated throughout the entity. ■ All – All operations ■ Persist ■ Merge ■ Move ■ Remove ■ Refresh		One-to-one mapping, One-to-many mapping, Many-to-many mapping, Many-to-one mapping
Mapped By	The field in the database table that "owns" the relationship. This field is required only on the non-owning side of the relationship.		One-to-one mapping, One-to-many mapping
Order By	Specify the default order for objects returned from a query: ■ No ordering ■ Primary key ■ Custom ordering This field corresponds to the `@OrderBy` annotation.	Primary key	One-to-many mapping. Many-to-many mapping, Many-to-one mapping

Property	Description	Default	Available for Mapping Type
Attribute Overrides	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).		Embedded mapping Embedded mapping

Related tasks

Mapping an entity

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.

Property	Description	Default
Name	Name of the join table that contains the foreign key column.	By default, the name is assumed to be the primary tables associated with the entities concatenated with an underscore.
Join Columns	Specify a mapped column for joining an entity association. This field corresponds to the `@JoinColum` attribute. Select **Override Default**, then Add, Edit, or Remove the join columns.	By default, the mapping is assumed to have a single join.
Inverse Join Columns	Select **Override Default**, then Add, Edit, or Remove the join columns.	

Related tasks

Mapping an entity

Related reference

Edit Join Columns Dialog

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.

Property	Description	Default
Join Column	Specify a mapped column for joining an entity association. This field corresponds to the `@JoinColum` attribute. Select **Override Default**, then Add, Edit, or Remove the join columns.	By default, the mapping is assumed to have a single join.

Related tasks

Mapping an entity

Related reference

Edit Join Columns Dialog

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.

Property	Description	Default
Primary Key Generation	These fields define how the primary key is generated. These fields correspond to the `@GeneratedValue` annotation.	Generated Value
Strategy	▪ Auto ▪ Identity – Values are assigned by the database's **Identity** column. ▪ Sequence – Values are assigned by a sequence table (see Sequence Generator). ▪ Table – Values are assigned by a database table (see Table Generator).	Auto
Generator Name	Unique name of the generated value.	
Table Generator	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**.	
Name	Unique name of the generator.	
Table	Database table that stores the generated ID values.	
Primary Key Column	The column in the table generator's **Table** that contains the primary key.	
Value Column	The column that stores the generated ID values.	

Property	Description	Default
Primary Key Column Value	The value for the **Primary Key Column** in the generator table.	
Sequence Generator	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**.	
Name	Name of the sequence table to use for defining primary key values.	
Sequence	Unique name of the sequence.	

Related tasks

ID mapping

Related reference

JPA Details view (for attributes)

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

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.

Property	Description	Default
Package	The Java package that contains the persistent entities. Click **Browse** and select the package	
Schema	The database schema that contains the **Table**. This field corresponds to the `<schema>` element in the `orm.xml` file.	
Catalog	The database catalog that contains the **Table**. This field corresponds to the `<catalog>` element in the `orm.xml` file.	

Property	Description	Default
Access	Specify the default access method for the variables in the project: ▪ Property ▪ Field This field corresponds to the `<access>` element in the `orm.xml` file.	

Related tasks

Adding persistence to a class

Related reference

JPA Details view (for entities)

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.

Property	Description	Default
XML Mapping Data Complete	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 `<xml-mapping-metadata-complete>` element in the `orm.xml` file.	
Schema	The database schema that contains the **Table**. This field corresponds to the `<schema>` element in the `orm.xml` file.	
Catalog	The database catalog that contains the **Table**. This field corresponds to the `<catalog>` element in the `orm.xml` file.	
Access Type	Specify how the entity its access instance variables. ▪ Property – Persistent state accessed through the property accessor methods. The property accessor methods must be **public** or **private**. ▪ Field – Instance variables are accessed directly. All non-transient instance variables are persistent.	Property
Cascade Persist	Adds cascade-persist to the set of cascade options in entity relationships of the persistence unit. This field corresponds to the `<cascade-persist>` element in the `orm.xml` file.	

Related tasks

Adding persistence to a class

Related reference

JPA Details view (for entities)

4.2.3.3 Generators

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

Property	Description
Generator	Displays the existing Sequence and Table generators. Click **Add Sequence** or **Add Table** to add a new generator.

Related tasks

Adding persistence to a class

Related reference

JPA Details view (for orm.xml)

4.2.3.4 Queries

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

Property	Description
Queries	Displays the existing Named and Native queries. Click **Add** to add a named query, or **Add Native** for a native query.

Related tasks

Creating Named Queries

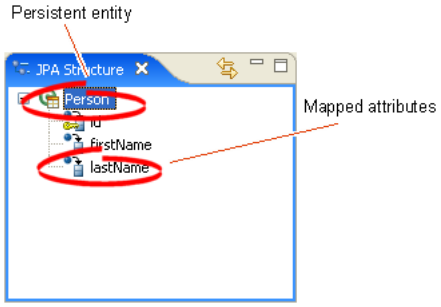
Related reference

JPA Details view (for orm.xml)

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



Related reference

JPA Development perspective

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 perisistence.xml Editor consists of the following pages:

- General
- Connection
- Customization
- Caching
- Logging
- Options
- Schema Generation
- Properties
- Source

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 4–4 Properties of the General Page

Property	Description	Default
Name	Enter the name of the persistence unit.	The project name.

Table 4–4 (Cont.) Properties of the General Page

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

4.2.5.2 Connection

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

Table 4–5 Properties of the Connection Page

Property	Description	Default
Transaction Type	Specify if the connection for this persistence unit uses one of the following transaction types: ■ **Default** -- Select to use the container used by the container. ■ **JTA** (Java Transaction API) -- Transactions of the Java EE server. ■ **Resource Local** -- Native actions of a JDBC driver that are referenced by a persistence unit.	
Batch Writing	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: ■ **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).	None
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.	

Table 4–5 (Cont.) Properties of the Connection Page

Property	Description	Default
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 note 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 property 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 property 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 4–6 Properties of the Customization Page

Property	Description	Default
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.	
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.	
Weaving Change Tracking	Select this option to use weaving to detect which fields or properties of the object change.	
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.	
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.	
Descriptor Customizer	Select an EclipseLink descriptor customizer class–a Java class that implements the `eclipselink.tools.sessionconfiguration.DescriptorCustomizer` interface and provides a default (zero-argument) constructor. Use this class's `customize` method, which takes an `eclipselink.descriptors.ClassDescriptor`, to programmatically access advanced EclipseLink descriptor and mapping API for the descriptor associated with the JPA entity named `<ENTITY>`.	

The following table lists the properties of the persistence.xml Editor.

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 4–7 Properties of the Caching Page

Property	Description	Default
Default Cache Type	Select one of the following as the Default Cache Type: ■ **Soft with Weak Subcache**–This option is similar to **Weak with Hard Subcache** 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. ■ **Week with Hard Subcache**–This option is similar to **Soft with Weak** subcache 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. ■ **Weak**–This option is similar to **Full**, except that objects are referenced using weak references. This option uses less memory than **Full**, 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. ■ **Soft**–This option is similar to **Weak** 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. ■ **Full**–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. ■ **None**–This option does not preserve object identity and does not cache objects.This option is not recommended.	Weak with hard subcache
Default Cache Size	Set the size of the cache.	100
Default Shared Cache	Specifies if cached instances should be in the shared cache or in a client isolated cache.	True

Table 4–7 (Cont.) Properties of the Caching Page

Property	Description	Default
Entity Caching	Specify the entity.	
Cache Type	Select a cache type. See *Default Cache*	
Cache Size	Set the size of the cache.	
Shared Cache	See *Default Shared Cache*.	

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

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 4–8 Properties of the Logging Page

Property	Description	Default
Logging Level	Specifies the amount and detail of log output by selecting the log level (in ascending order of information): The following are the valid values for the `java.util.logging.Level`: ■ **OFF**–disables logging ■ **SEVERE**–logs exceptions indicating TopLink cannot continue, as well as any exceptions generated during login. This includes a stack trace. ■ **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. ■ **INFO**–logs the login/logout per sever session, including the user name. After acquiring the session, detailed information is logged. ■ **CONFIG**–logs only login, JDBC connection, and database information. ■ **FINE**–logs SQL. ■ **FINER**–similar to warning. Includes stack trace. ■ **FINEST**–includes additional low level information. **Example**: `persistence.xml` file `<property name="eclipselink.logging.level" value="INFO"/>`	Info

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

Property	Description	Default
TimeStamp	Control whether the timestamp is logged in each log entry. The following are the valid values: ■ **true**–log a timestamp. ■ **false**–do not log a timestamp. **Example**: `persistence.xml` file `<property name="eclipselink.logging.timestamp" value="false"/>`	true
Thread	Control whether a thread identifier is logged in each log entry. The following are the valid values: ■ **true**–log a thread identifier. ■ **false**–do not log a thread identifier.	true
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 `<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 `<property name="eclipselink.logging.exceptions" value="true"/>`	false

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

Property	Description	Default
Logger	Select the type of logger to use: The following are the valid values: ■ **DefaultLogger**–the EclipseLink native logger `eclipselink.logging.DefaultSessionLog`. ■ **JavaLogger**–the `java.util.logging` logger `eclipselink.logging.JavaLog`. ■ **ServerLogger**–the `java.util.logging` logger `eclipselink.platform.server.ServerLog`. Integrates with the application server's logging as define in the `eclipselink.platform.server.ServerPlatform`. ■ Fully qualified class name of a custom logger. The custom logger must implement the `eclipselink.logging.SessionLog` interface. **Example**: `persistence.xml` file `<property name="eclipselink.logging.logger" value="acme.loggers.MyCustomLogger" />`	DefaultLogger

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 4–9 Properties of the Options Page

Property	Description	Default
Session Name	Specify the name by which the EclipseLink session is stored in the static session manager. Use this option if you need to access the EclipseLink shared session outside of the context of the JPA or to use a pre-existing EclipseLink session configured through a EclipseLink `sessions.xml` file Valid values: a valid EclipseLink session name that is unique in a server deployment. **Example**: `persistence.xml` file `<property name="eclipselink.session-name" value="MySession"/>`	

Table 4–9 (Cont.) Properties of the Options Page

Property	Description	Default
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 `<property name="toplink.session-xml" value="mysession.xml"/>`	
Target Database	Select the target database.	Auto
Target Server	Select the target server.	None
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 `<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.	

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 4–10

Property	Description	Default
DDL Generation Type	Select the type of DDL generation: ■ **None** -- Do not generate DDL; no schema is generated. ■ **Create Tables** -- Create DDL for non-existent tables; leave existing tables unchanged. ■ **Drop and Create Tables** -- Create DDL for all tables; drop all existing tables.	None
Output Mode	Select the DDL generation target: ■ **Both** -- Generate SQL files and execute them on the database. ■ **Database** -- Execute SQL on the database only (do not generate SQL files). ■ **SQL Script** -- Generate SQL files only (do not execute them on the database).	
DDL Generation Location	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.	
Create DDL File Name	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.	createDDL.jdbc
Drop DDL File Name	Specify the file name of the DDL file that EclipseLink generates that contains SQL statements for dropping tables for JPA entities.	dropDDL.jdbc

4.2.5.8 Properties

This page enables you to add or remove the vendor-specific `<properties>` elements of persistence.xml.

4.2.5.9 Source

Using this page, you can manually edit the persistence.xml file.

See "Managing the persistence.xml file" on page 3-7 for additional information.

Related tasks

Managing the persistence.xml file

4.3 Preferences

This section includes information on the following preference pages:

- Project Properties page – JPA Options

4.3.1 Project Properties page – JPA Options

Use the JPA 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.

Property	Description
Platform	Select the vendor-specific platform.
Connection	The database connection used to map the persistent entities. ■ To create a new connection, click **Add Connections**. ■ To reconnect to an existing connection, click **Reconnect**.
Override default schema from connection	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.
Persistent Class Management	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.

See "Modifying persistent project properties" on page 3-37 for additional information.

Related tasks

Modifying persistent project properties

4.4 Dialogs

This section includes information on the following preference pages:

- Generate Entities from Tables dialog
- Edit Join Columns Dialog

4.4.1 Generate Entities from Tables dialog

Use the Generate Entities dialog to create Java persistent entities from your database tables and columns.

This table lists the properties available in the Generate Entities dialog.

Property	Description
Source Folder	Enter a project folder name in which to generate the Java persistent entities, or click **Browse** to select an existing folder.
Package	Enter a package name in which to generate the Java persistent entities, or click **Browse** to select an existing package.
Synchronize Classes in persistence.xml	Specify if Dali should update the persistence.xml file to include the generated classes.

Property	Description
Tables	Select the tables from which to create Java persistent entities. The tables shown are determined by the database connection that you defined in the Project Properties page – JPA Options.

See "Generating entities from tables" on page 3-32 for more information.

Related tasks

Generating entities from tables

4.4.2 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.

Property	Description
Name	Name of the joint table column that contains the foreign key column.
Referenced Column Name	Name of the database column that contains the foreign key reference for the entity relationship.

Related reference

Join Table Information
Join Columns Information

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)

Related concepts

Perspectives

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.

Icon	Description
	Entity
	Embeddable entity
	Mapped superclass
	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

Related concepts

Icons and buttons

4.6.2 Buttons

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

Icon	Description
	JPA Development perspective

Related concepts

Icons and buttons

4.7 Dali Developer Documentation

Additional Dali documentation is available online at:

http://wiki.eclipse.org/index.php/Dali_Developer_Documentation

This developer documentation includes information about:

- Dali architecture
- Plugins that comprise the Dali JPA Eclipse feature
- Extension points

5

Tips and tricks

The following tips and tricks give some helpful ideas for increasing your productivity.

Tip	Description
Database Connections	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.
Schema-based persistence.xml	If you are behind a firewall, you may need to configure your Eclipse workspace proxy in the Preferences dialog (**Preferences > Internet > Proxy Settings**) to properly validate a schema-based `persistence.xml` file.

6

What's new

This section contains descriptions of the following new features and significant changes made to the Dali OR Mapping Tool for Release 2.1:

6.1 EclipseLink Support

Release 2.1 provides support of the following EclipseLink features.

- EclipseLink caching
- EclipseLink entity and attribute mapping options

  The Dali OR Mapping Tool can now configure Entities as Read-only, specify a customizer class, and configure change tracking.
- EclipseLink converters

EclipseLink (the Eclipse Persistence Services Project) is a complete persistence frame work. Refer to `http://www.eclipse.org/eclipselink/` for more information.

6.2 Multiple Mapping Files

The Dali OR Mapping Tool now supports configurations with multiple mapping (`orm.xml`) files. You can also create custom named mapping files.

7

Legal

Terms and conditions regarding the use of this guide.

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.

Index

Annotations

A

B

C

D

E

F

G

I

J

L

M

N

O

P

Q

R

S

T

V

W

X