**Using EclipseLink JPA Extensions (ELUG)**

Sommaire

[**Using EclipseLink JPA Extensions (ELUG)** 1](#_Toc303009062)

[**From Eclipsepedia** 3](#_Toc303009063)

[**Using EclipseLink JPA Extensions for Mapping** 4](#_Toc303009064)

[**How to Use the @BasicCollection Annotation** 5](#_Toc303009065)

[**How to Use the @BasicMap Annotation** 7](#_Toc303009066)

[**How to Use the @CollectionTable Annotation** 9](#_Toc303009067)

[**How to Use the @PrivateOwned Annotation** 11](#_Toc303009068)

[**How to Use the @JoinFetch Annotation** 13](#_Toc303009069)

[**How to Use the @Mutable Annotation** 15](#_Toc303009070)

[**How to Use the @Transformation Annotation** 16](#_Toc303009071)

[**How to Use the @ReadTransformer Annotation** 17](#_Toc303009072)

[**How to Use the @WriteTransformer Annotation** 18](#_Toc303009073)

[**How to Use the @WriteTransformers Annotation** 20](#_Toc303009074)

[**How to Use the @VariableOneToOne Annotation** 20](#_Toc303009075)

[**How to Use the Persistence Unit Properties for Mappings** 22](#_Toc303009076)

[**Using EclipseLink JPA Converters** 23](#_Toc303009077)

[**How to Use the @Converter Annotation** 24](#_Toc303009078)

[**How to Use the @TypeConverter Annotation** 25](#_Toc303009079)

[**How to Use the @ObjectTypeConverter Annotation** 27](#_Toc303009080)

[**How to Use the @StructConverter Annotation** 28](#_Toc303009081)

[**How to Use the @Convert Annotation** 30](#_Toc303009082)

[**Using EclipseLink JPA Extensions for Entity Caching** 32](#_Toc303009083)

[**How to Use the @Cache Annotation** 32](#_Toc303009084)

[**How to Use the Persistence Unit Properties for Caching** 39](#_Toc303009085)

[**How to Use the @TimeOfDay Annotation** 45](#_Toc303009086)

[**How to Use the @ExistenceChecking Annotation** 46](#_Toc303009087)

[**Using EclipseLink JPA Extensions for Customization and Optimization** 48](#_Toc303009088)

[**How to Use the @Customizer Annotation** 48](#_Toc303009089)

[**How to Use the Persistence Unit Properties for Customization and Validation** 49](#_Toc303009090)

[**How to Use the Persistence Unit Properties for Optimization** 58](#_Toc303009091)

[**Using EclipseLink JPA Extensions for Copy Policy** 61](#_Toc303009092)

[**How to Use the @CopyPolicy Annotation** 61](#_Toc303009093)

[**How to Use the @CloneCopyPolicy Annotation** 62](#_Toc303009094)

[**How to Use the @InstantiationCopyPolicy Annotation** 64](#_Toc303009095)

[**Using EclipseLink JPA Extensions for Declaration of Read-Only Classes** 64](#_Toc303009096)

[**How to Use the @ReadOnly Annotation** 64](#_Toc303009097)

[**Using EclipseLink JPA Extensions for Returning Policy** 65](#_Toc303009098)

[**How to Use the @ReturnInsert Annotation** 66](#_Toc303009099)

[**How to Use the @ReturnUpdate Annotation** 67](#_Toc303009100)

[**Using EclipseLink JPA Extensions for Optimistic Locking** 67](#_Toc303009101)

[**How to Use the @OptimisticLocking Annotation** 67](#_Toc303009102)

[**Using EclipseLink JPA Extensions for Stored Procedure Query** 72](#_Toc303009103)

[**How to Use the @NamedStoredProcedureQuery Annotation** 72](#_Toc303009104)

[**How to Use the @StoredProcedureParameter Annotation** 74](#_Toc303009105)

[**How to Use the @NamedStoredProcedureQueries Annotation** 75](#_Toc303009106)

[**Using EclipseLink JPA Extensions for JDBC** 76](#_Toc303009107)

[**How to Use EclipseLink JPA Extensions for JDBC Connection Communication** 76](#_Toc303009108)

[**How to Use EclipseLink JPA Extensions for JDBC Connection Pooling** 83](#_Toc303009109)

[**Using EclipseLink JPA Extensions for Logging** 85](#_Toc303009110)

[**Using EclipseLink JPA Extensions for Session, Target Database and Target Application Server** 90](#_Toc303009111)

[**Using EclipseLink JPA Extensions for Schema Generation** 96](#_Toc303009112)

[**Using EclipseLink JPA Extensions for Tracking Changes** 102](#_Toc303009113)

[**How to Use the @ChangeTracking Annotation** 102](#_Toc303009114)

[**How to Use the Persistence Unit Properties for Change Tracking** 105](#_Toc303009115)

[**What You May Need to Know About the Relationship Between the Change Tracking Annotation and Persistence Unit Property** 107](#_Toc303009116)

[**Using EclipseLink JPA Query Customization Extensions** 108](#_Toc303009117)

[**How to Use EclipseLink JPA Query Hints** 108](#_Toc303009118)

[**How to Use EclipseLink Query API in JPA Queries** 118](#_Toc303009119)

[**Using EclipseLink JPA Weaving** 123](#_Toc303009120)

[**How to Configure Dynamic Weaving for JPA Entities Using the EclipseLink Agent** 124](#_Toc303009121)

[**How to Configure Static Weaving for JPA Entities** 125](#_Toc303009122)

[**How to Disable Weaving Using EclipseLink Persistence Unit Properties** 130](#_Toc303009123)

[**What You May Need to Know About Weaving JPA Entities** 132](#_Toc303009124)

[**What You May Need to Know About EclipseLink JPA Lazy Loading** 132](#_Toc303009125)

[**What You May Need to Know About Overriding Annotations in JPA** 135](#_Toc303009126)

[**Overriding Annotations with eclipselink-orm.xml File** 135](#_Toc303009127)

[**Overriding Annotations with XML** 135](#_Toc303009128)

[**What You May Need to Know About EclipseLink JPA Overriding Mechanisms** 138](#_Toc303009129)

[**What You May Need to Know About Using EclipseLink JPA Persistence Unit Properties** 141](#_Toc303009130)

[**Allowing Zero Value Primary Keys** 144](#_Toc303009131)

[**Avoiding Cleartext Passwords** 144](#_Toc303009132)

**From Eclipsepedia**

[Related Topics](http://wiki.eclipse.org/Special:Whatlinkshere/Using_EclipseLink_JPA_Extensions_%28ELUG%29)

The Java Persistence API (JPA), part of the Java Enterprise Edition 5 (Java EE 5) EJB 3.0 specification, greatly simplifies Java persistence and provides an object relational mapping approach that allows you to declaratively define how to map Java objects to relational database tables in a standard, portable way that works both inside a Java EE 5 application server and outside an EJB container in a Java Standard Edition (Java SE) 5 application.

EclipseLink JPA provides extensions to what is defined in the JPA specification. These extensions come in persistence unit properties, query hints, annotations, EclipseLink own XML metadata, and custom API.

This section explains where and how you use the extensions to customize JPA behavior to meet your application requirements.

For more information, see the following:

* *EclipseLink API Reference*
* [*JSR-220 Enterprise JavaBeans v.3.0*](http://jcp.org/aboutJava/communityprocess/pfd/jsr220/indexl) *Java Persistence API specification*
* [*Java EE 5 SDK JPA Javadoc*](http://java.sun.com/javaee/5/docs/api/index.html?javax/persistence/package-summary.html)

**Using EclipseLink JPA Extensions for Mapping**

EclipseLink defines the following mapping metadata annotations (in addition to JPA-defined ones):

* [@BasicCollection](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40BasicCollection_Annotation)
* [@BasicMap](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40BasicMap_Annotation)
* [@CollectionTable](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40CollectionTable_Annotation)
* [@PrivateOwned](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40PrivateOwned_Annotation)
* [@JoinFetch](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40JoinFetch_Annotation)
* [@Mutable](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Mutable_Annotation)
* [@Transformation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Transformation_Annotation)
* [@ReadTransformer](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40ReadTransformer_Annotation)
* [@WriteTransformer](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40WriteTransformer_Annotation)
* [@WriteTransformers](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40WriteTransformers_Annotation)
* [@VariableOneToOne](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40VariableOneToOne_Annotation)

EclipseLink persistence provider searches mapping annotations in the following order:

* [@BasicCollection](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40BasicCollection_Annotation)
* [@BasicMap](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40BasicMap_Annotation)
* [@EmbeddedId](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40EmbeddedId)
* [@Embedded](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Embedded.29)
* [@ManyToMany](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40ManyToMany)
* [@ManyToOne](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40ManyToOne)
* [@OneToMany](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40OneToMany)
* [@OneToOne](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40OneToOne)

EclipseLink persistence provider applies the first annotation that it finds; it ignores other mapping annotations, if specified. In most cases, EclipseLink does not log warnings or throw exceptions for duplicate or incompatible mapping annotations.

If EclipseLink persistence provider does not find any of the mapping annotations from the preceding list, it applies the defaults defined by the JPA specification: not necessarily the [@Basic](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Basic) annotation.

**How to Use the @BasicCollection Annotation**

You can use the @BasicCollection annotation to map an org.eclipse.persistence.mappings.DirectCollectionMapping, which stores a collection of simple types, such asString, Number, Date, and so on, in a single table. The table must store the value and the foreign key to the source object.

@Target({METHOD, FIELD})

@Retention(RUNTIME)

public @interface BasicCollection {

FetchType fetch() default LAZY;

Column valueColumn() default @Column;

}

Use the @BasicCollection annotation in conjunction with a [@CollectionTable](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40CollectionTable_Annotation) annotation. You can also use it in conjunction with [@Convert](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Convert_Annotation) to modify the data value(s) during reading and writing of the collection, as well as with the [@JoinFetch](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40JoinFetch_Annotation) annotation.

This table lists attributes of the @BasicCollection annotation.

***Attributes of the @BasicCollection Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| fetch | The javax.persistence.FetchType enumerated type that defines whether EclipseLink should lazily load or eagerly fetch the value of the field or property. | FetchType.LAZY | optional |
| valueColumn | The name of the value column (javax.persistence.Column) that holds the direct collection data. | @ColumnNote: EclipseLink persistence provider sets the default to the name of the field or property. | optional |

|  |
| --- |
| **Note:** If you specify @BasicCollection on an attribute of type Map, EclipseLink will throw an exception: the type must be Collection, Set orList. If you specify the fetch type as LAZY, Collection implementation classes will also not be valid. |

This example shows how to use the @BasicCollection annotation to specify Employee field responsibilities.

***Usage of the @BasicCollection Annotation***

@Entity

**public** **class** Employee **implements** Serializable{

...

@BasicCollection (

fetch=FetchType.EAGER,

valueColumn=@Column(name="DESCRIPTION")

)

@CollectionTable (

name="RESPONS",

primaryKeyJoinColumns=

{@PrimaryKeyJoinColumn(name="EMPLOYEE\_ID", referencedColumnName="EMP\_ID")}

)

**public** Collection getResponsibilities() {

**return** responsibilities;

}

...

}

To further customize your mapping, use [the DirectCollectionMapping API](http://wiki.eclipse.org/Configuring_a_Relational_Direct_Collection_Mapping_%28ELUG%29).

**How to Use the @BasicMap Annotation**

You can use the @BasicMap annotation to map an [org.eclipse.persistence.mappings.DirectMapMapping](http://wiki.eclipse.org/Introduction_to_Relational_Mappings_%28ELUG%29#Direct_Map_Mapping), which stores a collection of key-value pairs of simple types, such asString, Number, Date, and so on, in a single table. The table must store the value and the foreign key to the source object.

@Target({METHOD, FIELD})

@Retention(RUNTIME)

public @interface BasicMap {

FetchType fetch() default LAZY;

Column keyColumn();

Convert keyConverter() default @Convert;

Column valueColumn() default @Column;

Convert valueConverter() default @Convert;

}

Use the @BasicMap annotation in conjunction with a [@CollectionTable](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40CollectionTable_Annotation) annotation, as well as with the [@JoinFetch](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40JoinFetch_Annotation) annotation.

This table lists attributes of the @BasicMap annotation.

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| fetch | Set this attribute to the javax.persistence.FetchType enumerated type to define whether EclipseLink persistence provider has to lazily load or eagerly fetch the value of the field or property. | FetchType.LAZY | optional |
| keyColumn | Set this attribute to the name of the data column (javax.persistence.Column) that holds the direct map key. | <field-name>\_KEY or <property-name>\_KEY (in uppercase characters) | optional |
| keyConverter | Set this attribute to the key converter ([@Convert](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Convert_Annotation)). | @Convert – an equivalent of specifying@Convert("none") resulting in no converter added to the direct map key. | optional |
| valueColumn | Set this attribute to the name of the value column (javax.persistence.Column) that holds the direct collection data. | @Column Field or property. | optional |
| valueConverter | Set this attribute to the value converter ([@Convert](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Convert_Annotation)). | @Convert – an equivalent of specifying@Convert("none") resulting in no converter added to the direct map key. | optional |

|  |
| --- |
| **Note:** If you specify @BasicMap on an attribute of type Collection, EclipseLink will throw an exception: the type must be Map. If you specify the fetch type as LAZY, Map implementation classes are also not valid. |

This example shows how to use the @BasicMap annotation to specify Employee field licenses.

***Usage of the @BasicMap Annotation***

@Entity

@Table(name="CMP3\_EMPLOYEE")

@TypeConverter(

name="Integer2String",

dataType=Integer.**class**,

objectType=String.**class**

)

**public** **class** Employee **implements** Serializable{

...

@BasicMap (

fetch=FetchType.EAGER,

keyColumn=@Column(name="LICENSE"),

keyConverter=@Convert("licenseConverter"),

valueColumn=@Column(name="STATUS"),

valueConverter=@Convert("Integer2String")

)

@ObjectTypeConverter(

name="licenseConverter",

conversionValues={

@ConversionValue(dataValue="AL", objectValue="Alcohol License"),

@ConversionValue(dataValue="FD", objectValue="Food License"),

@ConversionValue(dataValue="SM", objectValue="Smoking License"),

@ConversionValue(dataValue="SL", objectValue="Site Licence")}

)

@CollectionTable (

name="LICENSE",

primaryKeyJoinColumns={@PrimaryKeyJoinColumn(name="REST\_ID")}

)

**public** Map<String, String> getLicenses() {

**return** licenses;

}

...

}

To further customize your mapping, use the [DirectMapMapping API](http://wiki.eclipse.org/Configuring_a_Relational_Direct_Map_Mapping_%28ELUG%29).

**How to Use the @CollectionTable Annotation**

You can use the @CollectionTable annotation in conjunction with a [@BasicCollection](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40BasicCollection_Annotation) annotation or the [@BasicMap](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40BasicMap_Annotation) annotation. If you do not specify the @CollectionTable, EclipseLink persistence provider will use the defaults.

@Target({METHOD, FIELD})

@Retention(RUNTIME)

public @interface CollectionTable {

String name() default "";

String catalog() default "";

String schema() default "";

PrimaryKeyJoinColumn[] primaryKeyJoinColumns() default {};

UniqueConstraint[] uniqueConstraints() default {};

}

This table lists attributes of the @CollectionTable annotation.

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| name | Set this attribute to the String name for your collection table. | empty String  Note: EclipseLink persistence provider applies the following concatenatedString as a default: the name of the source entity + "\_" + the name of the relationship property or field of the source entity. | optional |
| catalog | Set this attribute to the String name of the table's catalog. | empty String Note: EclipseLink persistence provider sets the default to the persistence unit's default catalog. | optional |
| schema | Set this attribute to the String name of the table's schema. | empty String Note: EclipseLink persistence provider sets the default to the persistence unit's default schema. | optional |
| primaryKeyJoinColumns | Set this attribute to an array of javax.persistence.PrimaryKeyJoinColumn instances to specify a primary key column that is used as a foreign key to join to another table. If the source entity uses a composite primary key, you must specify a primary key join column for each field of the composite primary key. If the source entity uses a single primary key, you may choose to specify a primary key join column (optional). Otherwise, EclipseLink persistence provider will apply the following defaults:   * javax.persistence.PrimaryKeyJoinColumn name – the same name as the primary key column of the primary table of the source entity; * javax.persistence.PrimaryKeyJoinColumn referencedColumnName – the same name of the primary key column of the primary table of the source entity.   If the source entity uses a composite primary key and you failed to specify the primary key join columns, EclipseLink will throw an exception. | empty PrimaryKeyJoinColumn array | optional |
| uniqueConstraints | Set this attribute to an array of javax.persistence.UniqueConstraint instances that you want to place on the table. These constraints are only used if table generation is in effect. | empty UniqueConstraint array | optional |

This example shows how to use the @CollectionTable annotation to specify Employee field responsibilities.

***Usage of the @CollectionTable Annotation***

@Entity

**public** **class** Employee **implements** Serializable{

...

@BasicCollection (

fetch="LAZY",

valueColumn=@Column(name="DESCRIPTION")

)

@CollectionTable (

name="RESPONS",

primaryKeyJoinColumns=

{@PrimaryKeyJoinColumn(name="EMPLOYEE\_ID", referencedColumnName="EMP\_ID")}

)

**public** Collection getResponsibilities() {

**return** responsibilities;

}

...

}

**How to Use the @PrivateOwned Annotation**

Use the @PrivateOwned annotation in conjunction with a [@OneToOne](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40OneToOne) annotation, or a [@OneToMany](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40OneToMany) annotation.

@Target({METHOD, FIELD})

@Retention(RUNTIME)

public @interface PrivateOwned {}

The @PrivateOwned annotation does not have attributes.

This example shows how to use the @PrivateOwned annotation to specify Employee field phoneNumbers.

***Usage of the @PrivateOwned Annotation***

@Entity

**public** **class** Employee **implements** Serializable {

...

@OneToMany(cascade=ALL, mappedBy="employee")

@PrivateOwned

**public** Collection<PhoneNumber> getPhoneNumbers() {

**return** phoneNumbers;

}

...

}

**What You May Need to Know About Private Ownership of Objects**

When the referenced object is privately owned, the referenced child object cannot exist without the parent object.

When you tell EclipseLink that a relationship is privately owned, you are specifying the following:

* If the source of a privately owned relationship is deleted, then delete the target. Note that this is equivalent of setting a cascade delete.  
  For more information, see the following:
  + [Optimistic Version Locking Policies and Cascading](http://wiki.eclipse.org/Introduction_to_Descriptors_%28ELUG%29#Optimistic_Version_Locking_Policies_and_Cascading)
  + Section 3.2.2 "Removal" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220)
  + Section 3.5.2 "Semantics of the Life Cycle Callback Methods for Entities" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220)
  + Section 4.10 "Bulk Update and Delete Operations" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220)
  + [@OneToOne](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40OneToOne)
  + [@ManyToOne](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40ManyToOne)
  + [@OneToMany](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40OneToMany)
* If you remove the reference to a target from a source, then delete the target.

Do not configure privately owned relationships to objects that might be shared. An object should not be the target in more than one relationship if it is the target in a privately owned relationship.

The exception to this rule is the case when you have a many-to-many relationship in which a relation object is mapped to a relation table and is referenced through a one-to-many relationship by both the source and the target. In this case, if the one-to-many mapping is configured as privately owned, then when you delete the source, all the association objects will be deleted.

For more information, see [How to Use the privateOwnedRelationship Attribute](http://wiki.eclipse.org/Using_Basic_Unit_of_Work_API_%28ELUG%29#How_to_Use_the_privateOwnedRelationship_Attribute).

**How to Use the @JoinFetch Annotation**

You can specify the @JoinFetch annotation for the following mappings:

* [@OneToOne](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40OneToOne)
* [@OneToMany](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40OneToMany)
* [@ManyToOne](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40ManyToOne)
* [@ManyToMany](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40ManyToMany)
* [@BasicCollection](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40BasicCollection_Annotation)
* [@BasicMap](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40BasicMap_Annotation)

@Target({METHOD, FIELD})

@Retention(RUNTIME)

public @interface JoinFetch {

JoinFetchType value() default JoinFetchType.INNER;

}

Using the @JoinFetch annotation, you can enable the joining and reading of the related objects in the same query as the source object.

|  |
| --- |
| **Note:** We recommend setting join fetching at the query level, as not all queries require joining. For more information, see [Using Join Reading with ObjectLevelReadQuery](http://wiki.eclipse.org/Using_Basic_Query_API_%28ELUG%29#Using_Join_Reading_with_ObjectLevelReadQuery). |

Alternatively, you can use batch reading, especially for collection relationships. For more information, see [Using Batch Reading](http://wiki.eclipse.org/Using_Basic_Query_API_%28ELUG%29#Using_Batch_Reading).

This table lists attributes of the @JoinFetch annotation.

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| value | Set this attribute to the org.eclipse.persistence.annotations.JoinFetchType enumerated type of the fetch that you will be using.  The following are the valid values for the JoinFetchType:   * INNER – This option provides the inner join fetching of the related object. Note: Inner joining does not allow for null or empty values. * OUTER – This option provides the outer join fetching of the related object. Note: Outer joining allows for null or empty values.   For more information, see the following:   * [What You May Need to Know About Joins](http://wiki.eclipse.org/Introduction_to_EclipseLink_Expressions_%28ELUG%29#What_You_May_Need_to_Know_About_Joins) * [Using Join Reading with ObjectLevelReadQuery](http://wiki.eclipse.org/Using_Basic_Query_API_%28ELUG%29#Using_Join_Reading_with_ObjectLevelReadQuery) * [Configuring Joining at the Mapping Level](http://wiki.eclipse.org/Configuring_a_Relational_Mapping_%28ELUG%29#Configuring_Joining_at_the_Mapping_Level) | JoinFetchType.INNER | optional |

This example shows how to use the @JoinFetch annotation to specify Employee field managedEmployees.

***Usage of the @JoinFetch Annotation***

@Entity

**public** **class** Employee **implements** Serializable {

...

@OneToMany(cascade=ALL, mappedBy="owner")

@JoinFetch(value=OUTER)

**public** Collection<Employee> getManagedEmployees() {

**return** managedEmployees;

}

...

}

**How to Use the @Mutable Annotation**

You can specify the @Mutable annotation for the following mappings:

* [@Basic](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Basic)
* [@Id](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Id)
* [@Version](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#Configuring_Locking)
* [@Transformation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Transformation_Annotation)

@Target({METHOD, FIELD})

@Retention(RUNTIME)

public @interface Mutable {

boolean value() default true;

}

Using the @Mutable annotation, you can indicate that the value of a complex field itself can be changed or not changed (instead of being replaced).

By default, EclipseLink assumes that Serializable types are mutable. In other words, EclipseLink assumes the default @Mutable(value=true).

You can override this configuration using @Mutable(value=false).

By default, EclipseLink assumes that all @Basic mapping types, except Serializable types, are immutable.

You can override this configuration using @Mutable(value=true). For example, if you need to call Date or Calendar set methods, you can decorate a Date or Calendarpersistent field using @Mutable(value=true).

|  |
| --- |
| **Note:** For Date and Calendar types only, you can configure all such persistent fields as mutable by setting global persistence unit propertyeclipselink.temporal.mutable to true. For more information, see the [EclipseLink JPA Persistence Unit Properties for Mappings](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Table_19-6) table. |

Mutable basic mappings affect the overhead of change tracking. Attribute change tracking can only be weaved with immutable mappings.

For more information, see the following:

* [Unit of Work and Change Policy](http://wiki.eclipse.org/Introduction_to_EclipseLink_Transactions_%28ELUG%29#Unit_of_Work_and_Change_Policy)
* [Using Weaving](http://wiki.eclipse.org/Introduction_to_EclipseLink_Application_Development_%28ELUG%29#Using_Weaving)
* [Mutability](http://wiki.eclipse.org/Introduction_to_EclipseLink_Application_Development_%28ELUG%29#Mutability)

This table lists attributes of the @Mutable annotation.

***Attributes of the @Mutable Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| value | Set this attribute to one of the following boolean values:   * true – The object is mutable. * false – The object is immutable. | true | optional |

This example shows how to use the @Mutable annotation to specify Employee field hireDate.

***Usage of the @Mutable Annotation***

@Entity

**public** **class** Employee **implements** Serializable {

...

@Temporal(DATE)

@Mutable

**public** Calendar getHireDate() {

**return** hireDate;

}

...

}

**How to Use the @Transformation Annotation**

You can use the @Transformation annotation to map an [org.eclipse.persistence.mappings.TransformationMapping](http://wiki.eclipse.org/Introduction_to_Relational_Mappings_%28ELUG%29#Transformation_Mapping), which allows to map an attribute to one or more database columns.

@Target({METHOD, FIELD})

@Retention(RUNTIME)

public @interface Transformation {

FetchType fetch() default EAGER;

boolean optional() default true;

}

|  |
| --- |
| **Note:** The @Transformation annotation is optional for transformation mappings: if you specify either a [@ReadTransformer](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40ReadTransformer_Annotation),[@WriteTransformer](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40WriteTransformer_Annotation) or [@WriteTransformers](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40WriteTransformers_Annotation) annotation, the mapping would become a transformation mapping by default, without you specifying the @Transformation annotation. |

This table lists attributes of the @Transformation annotation.

***Attributes of the @Transformation Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| fetch | The javax.persistence.FetchType enumerated type that defines whether EclipseLink should lazily load or eagerly fetch the value of the field or property. | FetchType.EAGER | optional |
| optional | Set this attribute to define whether or not the value of the field or property may be null.  Note: the value you set is disregarded for primitive types, which are considered mandatory.  The following are valid values:   * true - The value of the field or property may be null. * false - The value of the field or property may not be null. | true | optional |

For more information, see the following:

* [How to Use the @ReadTransformer Annotation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40ReadTransformer_Annotation)
* [How to Use the @WriteTransformer Annotation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40WriteTransformer_Annotation)

**How to Use the @ReadTransformer Annotation**

Use the @ReadTransformer annotation with the [@Transformation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Transformation_Annotation) mapping to define transformation of one or more database column values into an attribute value. Note that you can only use this annotation if the [@Transformation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Transformation_Annotation) mapping is not write-only.

@Target(value={METHOD,FIELD})

@Retention(value=RUNTIME)

public @interface ReadTransformer {

Class transformerClass() default void.class;

String method() default "";

}

This table lists attributes of the @ReadTransformer annotation.

***Attributes of the @ReadTransformer Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| method | Set this attribute to the String method name that the mapped class must have. This method does not assign a value to the attribute; instead, it returns the value to be assigned to the attribute. | emptyString | required/optional1 |
| transformerClass | Set this attribute to the Class that implements theorg.eclipse.persistence.mappings.transformers.AttributeTransformer interface. This will instantiate the class and use its buildAttributeValue method to create the value to be assigned to the attribute.  For more information, see see [How to Configure Attribute Transformer Using Java](http://wiki.eclipse.org/Configuring_a_Mapping_%28ELUG%29#How_to_Configure_Attribute_Transformer_Using_Java). | void.class | required/optional1 |

1You must specify either the transformerClass or method, but not both

**How to Use the @WriteTransformer Annotation**

Use the @WriteTransformer annotation with the [@Transformation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Transformation_Annotation) mapping to define transformation of an attribute value to a single database column value. Note that you can only use this annotation if the [@Transformation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Transformation_Annotation) mapping is not read-only.

@Target(value={METHOD,FIELD})

@Retention(value=RUNTIME)

public @interface WriteTransformer {

Class transformerClass() default void.class;

String method() default "";

Column column() default @Column;

}

This table lists attributes of the @WriteTransformer annotation.

***Attributes of the @WriteTransformer Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| method | Set this attribute to the String method name that the mapped class must have. This method returns the value to be written into the database column.  Note: for proper support of [DDL generation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_EclipseLink_JPA_Extensions_for_Schema_Generation) and [returning policy](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_EclipseLink_JPA_Extensions_for_Returning_Policy), define the method to return not just an Object, but a particular type, as the following example shows:  public Time getStartTime()  The method may require a [@Transient](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Transient) annotation to avoid being mapped as the [@Basic](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Basic) by default. | empty String | required/optional1 |
| transformerClass | Set this attribute to the Class that implements theorg.eclipse.persistence.mappings.transformers.FieldTransformer interface. This will instantiate the class and use its buildFieldValue method to create the value to be written into the database column.  For more information, see see [How to Configure Field Transformer Associations](http://wiki.eclipse.org/Configuring_a_Mapping_%28ELUG%29#How_to_Configure_Field_Transformer_Associations_Using_Java).  Note: for proper support of [DDL generation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_EclipseLink_JPA_Extensions_for_Schema_Generation) and [returning policy](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_EclipseLink_JPA_Extensions_for_Returning_Policy), define the method to return not just an Object, but a relevant Java type, as the following example shows:  public Time buildFieldValue(Object instance, String fieldName, Session session) | void.class | required/optional1 |
| column | Set this attribute to a Column into which the value should be written.  You may choose not to set this attribute if a single WriteTransformer annotates an attribute. In this case, the attribute's name will be used as a column name. | @javax.persistence.Column (see Section 9.1.5 "Column Annotation" of the JPA specification) | required |

1You must specify either the transformerClass or method, but not both.

**How to Use the @WriteTransformers Annotation**

Use the @WriteTransformers annotation with the [@Transformation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Transformation_Annotation) mapping to wrap multiple [write transformers](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40WriteTransformers_Annotation). Note that you can only use this annotation if the[@Transformation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Transformation_Annotation) mapping is not read-only.

@Target(value={METHOD,FIELD})

@Retention(value=RUNTIME)

public @interface WriteTransformers {

WriteTransformer[] value();

}

This table lists attributes of the @WriteTransformers annotation.

***Attributes of the @WriteTransformers Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| value | Set this attribute to the array of [WriteTransformer](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40WriteTransformer_Annotation). | no default | optional |

**How to Use the @VariableOneToOne Annotation**

You can use the @VariableOneToOne annotation to map an [org.eclipse.persistence.mappings.VariableOneToOneMapping](http://wiki.eclipse.org/Introduction_to_Relational_Mappings_%28ELUG%29#Variable_One-to-One_Mapping), which you use to represent a pointer references between a Java object and an implementer of an interface. This mapping is typically represented by a single pointer (stored in an instance variable) between the source and target objects. In the relational database tables, these mappings are usually implemented using a foreign key and a type code.

@Target({METHOD, FIELD})

@Retention(RUNTIME)

public @interface VariableOneToOne {

Class targetInterface() default void.class;

CascadeType[] cascade() default {};

FetchType fetch() default EAGER;

boolean optional() default true;

DiscriminatorColumn discriminatorColumn() default @DiscriminatorColumn;

DiscriminatorClass[] discriminatorClasses() default {};

}

Specify the @VariableOneToOne annotation within an @Entity (see Section 8.1 "Entity" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220)), [@MappedSuperclass](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40MappedSuperclass) or [@Embeddable](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Embeddable) class.

This table lists attributes of the @VariableOneToOne annotation.

***Attributes of the @VariableOneToOneAnnotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| cascade | Set this attribute to an array of javax.persistence.CascadeType objects to indicate operations that must be cascaded to the target of the association. | {} | optional |
| discriminatorClasses | Set this attribute to an array of org.eclipse.persistence.annotations.DiscriminatorClassobjects to provide a list of discriminator types that can be used with this variable one-to-one mapping.  If none are specified, then those entities within the persistence unit that implement the target interface will be added to the list of types. In this case, the discriminator type will default as follows:   * If the javax.persistence.DiscriminatorColumn type is STRING, it is Entity.name() * If the DiscriminatorColumn type is CHAR, it is the first letter of the Entity class * If the DiscriminatorColumn type is INTEGER it is the next integer after the highest integer explicitly added. | {} | optional |
| discriminatorColumn | Set this attribute to the javax.persistence.DiscriminatorColumn that will hold the type indicators.  If the discriminator column is not specified, the name of the discriminator column defaults to"DTYPE", and the discriminator type - to STRING. | @javax.persistence.DiscriminatorColumn | optional |
| fetch | The javax.persistence.FetchType enumerated type that defines whether EclipseLink should lazily load or eagerly fetch the value of the field or property. | FetchType.EAGER | optional |
| optional | Set this attribute to define whether or not the association is optional.  The following are valid values:   * true - The association is optional. * false - A non-null relationship must always exist. | true | optional |
| targetInterface | Set this attribute to an interface class that is the target of the association.  If not specified, it will be inferred from the type of the object being referenced. | void.class | optional |

**How to Use the Persistence Unit Properties for Mappings**

This table lists the persistence unit properties that you can define in a persistence.xml file to configure EclipseLink mappings.

***EclipseLink JPA Persistence Unit Properties for Mappings***

|  |  |  |
| --- | --- | --- |
| **Property** | **Usage** | **Default** |
| eclipselink.temporal.mutable | Specify whether or not EclipseLink JPA should handle all Date and Calendar persistent fields as mutable objects.  The following are the valid values:   * true – all Date and Calendar persistent fields are mutable. * false – all Date and Calendar persistent fields are immutable.   For more information, see the following:   * [How to Use the @Mutable Annotation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Mutable_Annotation) * [Mutability](http://wiki.eclipse.org/Introduction_to_EclipseLink_Application_Development_%28ELUG%29#Mutability)   **Example**: persistence.xml file  <property name="eclipselink.temporal.mutable" value="true"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.TEMPORAL\_MUTABLE, "false"); | false |

For information about the use of annotations as opposed to persistence unit properties and vice versa, see the following:

* [What you May Need to Know About Overriding Annotations in JPA](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_you_May_Need_to_Know_About_Overriding_Annotations_in_JPA)
* [What You May Need to Know About EclipseLink JPA Overriding Mechanisms](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_you_May_Need_to_Know_About_EclipseLink_JPA_Overriding_Mechanisms_.28ELUG.29).

For more information about persistence unit properties, see [What you May Need to Know About Using EclipseLink JPA Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_you_May_Need_to_Know_About_Using_EclipseLink_JPA_Persistence_Unit_Properties).

**Using EclipseLink JPA Converters**

EclipseLink defines the following converter annotations (in addition to JPA-defined ones):

* [@Converter](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Converter_Annotation)
* [@TypeConverter](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40TypeConverter_Annotation)
* [@ObjectTypeConverter](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40ObjectTypeConverter_Annotation)
* [@StructConverter](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40StructConverter_Annotation)
* [@Convert](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Convert_Annotation)

EclipseLink persistence provider searches the converter annotations in the following order:

* [@Convert](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Convert_Annotation)
* [@Enumerated](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Enumerated)
* [@Lob](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Lob)
* [@Temporal](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Temporal)
* Serialized (automatic)

You can define converters at the class, field and property level. You can specify EclipseLink converters on the following classes:

* @Entity (see Section 8.1 "Entity" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220))
* [@MappedSuperclass](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40MappedSuperclass)
* [@Embeddable](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Embeddable)

You can use EclipseLink converters with the following mappings:

* [@Basic](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Basic)
* [@Id](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Id)
* [@Version](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#Configuring_Locking)
* [@BasicMap](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40BasicMap_Annotation)
* [@BasicCollection](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40BasicCollection_Annotation)

If you specify a converter with any other type of mapping annotation, EclipseLink will throw an exception.

**How to Use the @Converter Annotation**

You can use @Converter annotation to specify a custom converter for modification of the data value(s) during the reading and writing of a mapped attribute.

@Target({TYPE, METHOD, FIELD})

@Retention(RUNTIME)

public @interface Converter {

String name();

Class converterClass();

}

This table lists attributes of the @Converter annotation.

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| name | Set this attribute to the String name for your converter. Ensure that this name is unique across the persistence unit | no default | required |
| converterClass | Set this attribute to the Class of your converter. This class must implement the EclipseLinkorg.eclipse.persistence.mappings.converters.Converter interface. | no default | required |

This example shows how to use the @Converter annotation to specify Employee field gender.

***Usage of the @Converter Annotation***

@Entity

**public** **class** Employee **implements** Serializable{

...

@Basic

@Converter (

name="genderConverter",

converterClass=org.myorg.converters.GenderConverter.**class**

)

@Convert("genderConverter")

**public** String getGender() {

**return** gender;

}

...

}

**How to Use the @TypeConverter Annotation**

The @TypeConverter is an EclipseLink-specific annotation. You can use it to specify an org.eclipse.persistence.mappings.converters.TypeConversionConverter for modification of the data value(s) during the reading and writing of a mapped attribute.

@Target({TYPE, METHOD, FIELD})

@Retention(RUNTIME)

public @interface TypeConverter {

String name();

Class dataType() default void.class;

Class objectType() default void.class;

}

This table lists attributes of the @TypeConverter annotation.

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| name | Set this attribute to the String name for your converter. Ensure that this name is unique across the persistence unit. | no default | required |
| dataType | Set this attribute to the type stored in the database. | void.class1 | optional |
| objectType | Set the value of this attribute to the type stored on the entity. | void.class1 | optional |

1 The default is inferred from the type of the persistence field or property.

This example shows how to use the @TypeConverter annotation to convert the Double value stored in the database to a Float value stored in the entity.

***Usage of the @TypeConverter Annotation***

@Entity

**public** **class** Employee **implements** Serializable{

...

@TypeConverter (

name="doubleToFloat",

dataType=Double.**class**,

objectType=Float.**class**,

)

@Convert("doubleToFloat")

**public** Number getGradePointAverage() {

**return** gradePointAverage;

}

...

}

* Another example of **TypeConverter** usage is in the [distributed tutorial](http://wiki.eclipse.org/EclipseLink/Examples/Distributed#TypeConverter) - this one is used to enable storage of a near limitless **BigInteger** using a String (VARCHAR) field instead of the usual NUMBER type.

**How to Use the @ObjectTypeConverter Annotation**

You can use the @ObjectTypeConverter annotation to specify an org.eclipse.persistence.mappings.converters.ObjectTypeConverter that converts a fixed number of database data value(s) to Java object value(s) during the reading and writing of a mapped attribute.

@Target({TYPE, METHOD, FIELD})

@Retention(RUNTIME)

public @interface ObjectTypeConverter {

String name();

Class dataType() default void.class;

Class objectType() default void.class;

ConversionValue[] conversionValues();

String defaultObjectValue() default "";

}

This table lists attributes of the @ObjectTypeConverter annotation.

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| name | Set this attribute to the String name for your converter. Ensure that this name is unique across the persistence unit | no default | required |
| dataType | Set this attribute to the type stored in the database. | void.class1 | optional |
| objectType | Set the value of this attribute to the type stored on the entity. | void.class1 | optional |
| conversionValues | Set the value of this attribute to the array of conversion values (instances of ConversionValue: String objectValue and String dataValue. See the [Usage of the @ObjectTypeConverter Annotation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-9) example, to be used with the object converter. | no default | required |
| defaultObjectValue | Set the value of this attribute to the default object value. Note that this argument is for dealing with legacy data if the data value is missing. | emptyString | optional |

1 The default is inferred from the type of the persistence field or property.

This example shows how to use the @ObjectTypeConverter annotation to specify the Employee field gender.

***Usage of the @ObjectTypeConverter Annotation***

**public** **class** Employee **implements** Serializable{

...

@ObjectTypeConverter (

name="genderConverter",

dataType=java.lang.String.**class**,

objectType=java.lang.String.**class**,

conversionValues={

@ConversionValue(dataValue="F", objectValue="Female"),

@ConversionValue(dataValue="M", objectValue="Male")}

)

@Convert("genderConverter")

**public** String getGender() {

**return** gender;

}

...

}

**How to Use the @StructConverter Annotation**

The @StructConverter is an EclipseLink-specific annotation. You can add it to an org.eclipse.persistence.platform.database.DatabasePlatform using itsaddStructConverter method to enable custom processing of java.sql.Struct types.

@Target({TYPE, METHOD, FIELD})

@Retention(RUNTIME)

public @interface StructConverter {

String name();

String converter();

}

This table lists attributes of the @StructConverter annotation.

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| name | Set this attribute to the String name for your converter. Ensure that this name is unique across the persistence unit. | no default | required |
| converter | Set this attribute to the converter class as a String. This class must implement the EclipseLinkorg.eclipse.persistence.mappings.converters.Converter interface. | no default | required |

This example shows how to define the @StructConverter.

***Defining the @StructConverter***

@StructConverter(name="MyType",

converter="myproject.converters.MyStructConverter")

You can specify the @StructConverter annotation anywhere in an Entity with the scope being the whole session.

EclipseLink will throw an exception if you add more than one StructConverter that affects the same Java type.

A @StructConverter exists in the same namespaces as [@Converter](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Converter_Annotation). EclipseLink will throw a validation exception if you add a Converter and a StructConverter of the same name.

|  |
| --- |
| **Note:** You can also configure structure converters in a sessions.xml file (see [What You May Need to Know About EclipseLink JPA Overriding Mechanisms](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_EclipseLink_JPA_Overriding_Mechanisms)). |

**Using Structure Converters to Configure Mappings**

In EclipseLink, a DatabasePlatform (see [Database Platforms](http://wiki.eclipse.org/Introduction_to_Data_Access_%28ELUG%29#Database_Platforms)) holds a structure converter. An org.eclipse.persistence.database.platform.converters.StructConverteraffects all objects of a particular type read into the Session that has that DatabasePlatform. This prevents you from configuring the StructConverter on a mapping-by-mapping basis. To configure mappings that use the StructConverter, you call their setFieldType(java.sql.Types.STRUCT) method. You must call this method on all mappings that theStructConverter will affect – if you do not call it, errors might occur.

The JPA specification requires all @Basic mappings (see [@Basic](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Basic)) that map to a non-primitive or a non-primitive-wrapper type have a serialized converter added to them. This enables certain STRUCT types to map to a field without serialization.

You can use the existing [@Convert](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Convert_Annotation) annotation with its value attribute set to the StructConverter name – in this case, EclipseLink will apply appropriate settings to the mapping. This setting will be required on all mappings that use a type for which a StructConverter has been defined. Failing to configure the mapping with the @Convert will cause an error.

For more information, see the following:

* [Object-Relational Data Type Structure Mapping](http://wiki.eclipse.org/Introduction_to_Object-Relational_Data_Type_Mappings_%28ELUG%29#Object-Relational_Data_Type_Structure_Mapping)
* [Object-Relational Data Type Descriptors](http://wiki.eclipse.org/Introduction_to_Object-Relational_Data_Type_Descriptors_%28ELUG%29)

**How to Use the @Convert Annotation**

The @Convert annotation specifies that a named converter should be used with the corresponding mapped attribute.

@Target({METHOD, FIELD})

@Retention(RUNTIME)

public @interface Convert {

String value() default "none";

}

The @Convert has the following reserved names:

* serialized – places the org.eclipse.persistence.mappings.converters.SerializedObjectConverter on the associated mapping.
* none – does not place a converter on the associated mapping.

This table lists attributes of the @Convert annotation.

***Attributes of the @Convert Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| value | Set this attribute to the String name for your converter. | "none" String | optional |

This example shows how to use the @Convert annotation to define the Employee field gender.

***Usage of the @Convert Annotation***

@Entity

@Table(name="EMPLOYEE")

@Converter(

name="genderConverter",

converterClass=org.myorg.converters.GenderConverter.**class**

)

**public** **class** Employee **implements** Serializable{

...

@Basic

@Convert("genderConverter")

**public** String getGender() {

**return** gender;

}

...

}

**Using EclipseLink JPA Extensions for Entity Caching**

The EclipseLink cache is an in-memory repository that stores recently read or written objects based on class and primary key values. EclipseLink uses the cache to do the following:

* Improve performance by holding recently read or written objects and accessing them in-memory to minimize database access.
* Manage locking and isolation level.
* Manage object identity.

For more information about the EclipseLink cache and its default behavior, see [Introduction to Cache](http://wiki.eclipse.org/Introduction_to_Cache_%28ELUG%29).

EclipseLink defines the following entity caching annotations:

* [@Cache](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Cache_Annotation)
* [@TimeOfDay](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40TimeOfDay_Annotation)
* [@ExistenceChecking](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40ExistenceChecking_Annotation)

EclipseLink also provides a number of persistence unit properties that you can specify to configure the EclipseLink cache (see [How to Use the Persistence Unit Properties for Caching](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_Persistence_Unit_Properties_for_Caching)). These properties may compliment or provide an alternative to the usage of annotations.

For more information, see the following:

* [What You May Need to Know About Overriding Annotations in JPA](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Overriding_Annotations_in_JPA)
* [What You May Need to Know About Using EclipseLink JPA Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Using_EclipseLink_JPA_Persistence_Unit_Properties)

**How to Use the @Cache Annotation**

EclipseLink uses identity maps to cache objects in order to enhance performance, as well as maintain object identity. You can control the cache and its behavior by decorating your entity classes with the @Cache annotation.

@Target({TYPE})

@Retention(RUNTIME)

**public** @**interface** Cache {

CacheType type() **default** SOFT\_WEAK;

**int** size() **default** 100;

**boolean** shared() **default** **true**;

**int** expiry() **default** -1;

TimeOfDay expiryTimeOfDay() **default** @TimeOfDay(specified=**false**);

**boolean** alwaysRefresh() **default** **false**;

**boolean** refreshOnlyIfNewer() **default** **false**;

**boolean** disableHits() **default** **false**;

CacheCoordinationType coordinationType() **default** SEND\_OBJECT\_CHANGES;

}

You may define the @Cache annotation on the following:

* @Entity (see Section 8.1 "Entity" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220));
* [@MappedSuperclass](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40MappedSuperclass);
* the root of the inheritance hierarchy (if applicable).

|  |
| --- |
| **Note:** If you define the @Cache annotation on an inheritance subclass, the annotation will be ignored. |

***Attributes of the @Cache Annotation***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** | **Override with Persistence Unit Property** |
| type | Set this attribute to the type (org.eclipse.persistence.annotations.CacheType enumerated type) of the cache that you will be using.  The following are the valid values for the CacheType:   * 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. * 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. We recommend using 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. * SOFT\_WEAK – This option is similar to WEAK 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. We recommend using this identity map in most circumstances as a means to control memory used by the cache. * HARD\_WEAK – This option is similar to SOFT\_WEAK 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. * CACHE – With this option, a cache identity map maintains a fixed number of objects that you specify in your application. Objects are removed from the cache on a least-recently-used basis. This option allows object identity for the most commonly used objects.  Note: this option furnishes caching and identity, but does not guarantee identity. * NONE – This option does not preserve object identity and does not cache objects. We do not recommend using this option. To disable the shared cache for an Entity set the 'shared' property to false. | CacheType.SOFT\_WEAK | optional | * [eclipselink.cache.type.<ENTITY>](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.cache.type.ENTITY) * [eclipselink.cache.type.default](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.cache.type.default) |
| size | Set this attribute to an int value to define the size of cache to use (number of objects). | 100 |  |  |
| shared | Set this attribute to a boolean value to indicate whether cached instances should be in the shared cache or in a client isolated cache (see [Isolated Client Session Cache](http://wiki.eclipse.org/Using_Advanced_Unit_of_Work_API_%28ELUG%29#Isolated_Client_Session_Cache)).  The following are the valid values:   * true - use shared cache for cached instances; * false - use client isolated cache for cached instances. | true | optional | * [eclipselink.cache.shared.default](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.cache.shared.default) |
| expiry | Set this attribute to the int value to enable the expiration of the cached instance after a fixed period of time (milliseconds). Queries executed against the cache after this will be forced back to the database for a refreshed copy. | -1 (no expiry) | optional |  |
| expiryTimeOfDay | Set this attribute to a specific time of day (org.eclipse.persistence.annotations.TimeOfDay) when the cached instance will expire. Queries executed against the cache after this will be forced back to the database for a refreshed copy. | @TimeOfDay(specified=false) | optional |  |
| alwaysRefresh | Set this attribute to a boolean value of true to force all queries that go to the database to always refresh the cache. | false | optional |  |
| refreshOnlyIfNewer | Set this attribute to a boolean value of true to force all queries that go to the database to refresh the cache only if the data received from the database by a query is newer than the data in the cache (as determined by the optimistic locking field).  Note: This option only applies if one of the other refreshing options, such as alwaysRefresh, is already enabled.  Note: A version field is necessary to apply this feature.  For more information, see the following:   * [What You May Need to Know About Version Fields](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Version_Fields) * [Optimistic Version Locking Policies](http://wiki.eclipse.org/Introduction_to_Descriptors_%28ELUG%29#Optimistic_Version_Locking_Policies) * [Configuring Locking](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#Configuring_Locking) * Section 3.4 "Optimistic Locking and Concurrency" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220) * Section 9.1.17 "Version Annotation" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220) | false | optional |  |
| disableHits | Set this attribute to a boolean value of true to force all queries to bypass the cache for hits, but still resolve against the cache for identity. This forces all queries to hit the database. | false | optional |  |
| coordinationType | Set this attribute to the cache coordination mode (org.eclipse.persistence.annotations.CacheCoordinationTypeenumerated type).  The following are the valid values for the CacheCoordinationType:   * SEND\_OBJECT\_CHANGES – This option sends a list of changed objects (including information about the changes). This data is merged into the receiving cache. * INVALIDATE\_CHANGED\_OBJECTS – This option sends a list of the identities of the objects that have changed. The receiving cache invalidates the objects (rather than changing any of the data). * SEND\_NEW\_OBJECTS\_WITH\_CHANGES – This option is similar toSEND\_OBJECT\_CHANGES except it also includes any newly created objects from the transaction. * NONE – This option does not coordinate cache. For more information, see [Cache Coordination](http://wiki.eclipse.org/Introduction_to_Cache_%28ELUG%29#Cache_Coordination). | CacheCoordinationType.SEND\_OBJECT\_CHANGES | optional |  |

|  |
| --- |
| **Note:** If you define the @Cache annotation on @Embeddable (see [@Embeddable](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Embeddable)), EclipseLink will throw an exception. |

This example shows how to achieve the desired behavior of the EclipseLink cache by defining the attributes of the @Cache annotation.

***Usage of @Cache Annotation***

@Entity

@Table(name="EMPLOYEE")

@Cache (

type=CacheType.WEAK,

isolated=**false**,

expiry=600000,

alwaysRefresh=**true**,

disableHits=**true**,

coordinationType=INVALIDATE\_CHANGED\_OBJECTS

)

**public** **class** Employee **implements** Serializable {

...

}

**What You May Need to Know About Version Fields**

By default, EclipseLink persistence provider assumes that the application is responsible for data consistency.

Use the @Version annotation (see [Configuring Locking](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#Configuring_Locking)) to enable the JPA-managed optimistic locking by specifying the version field or property of an entity class that serves as its optimistic lock value (recommended).

When choosing a version field or property, ensure that the following is true:

* there is only one version field or property per entity;
* you choose a property or field persisted to the primary table (see Section 9.1.1 "Table Annotation" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220));
* your application does not modify the version property or field.

**How to Use the Persistence Unit Properties for Caching**

The [EclipseLink JPA Properties for Caching](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Table_19-13) table lists the persistence unit properties that you can define in a persistence.xml file to configure the EclipseLink cache.

For more information, see the following:

* [Introduction to Cache](http://wiki.eclipse.org/Introduction_to_Cache_%28ELUG%29)
* [What You May Need to Know About EclipseLink JPA Overriding Mechanisms](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_EclipseLink_JPA_Overriding_Mechanisms)
* [What You May Need to Know About Overriding Annotations in JPA](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Overriding_Annotations_in_JPA)
* [What You May Need to Know About Using EclipseLink JPA Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Using_EclipseLink_JPA_Persistence_Unit_Properties)

***EclipseLink JPA Properties for Caching***

|  |  |  |
| --- | --- | --- |
| **Property** | **Usage** | **Default** |
| eclipselink.cache.type.default | The default type of session cache.  A session cache is a shared cache that services clients attached to a given session. When you read objects from or write objects to the data source using a client session, EclipseLink saves a copy of the objects in the parent server session's cache and makes them accessible to child client sessions. From a JPA perspective, an EntityManagerFactory wraps anorg.eclipse.persistence.sessions.server.ServerSession; entity managers wrap anorg.eclipse.persistence.sessions.UnitOfWork andorg.eclipse.persistence.sessions.server.ClientSession. For more information about sessions, see [Introduction to EclipseLink Sessions](http://wiki.eclipse.org/Introduction_to_EclipseLink_Sessions_%28ELUG%29).  The following are the valid values for the use in a persistence.xml file and for theorg.eclipse.persistence.config.CacheType:   * Full – This option provides full caching and guaranteed identity: objects are never flushed from memory unless they are deleted. For more information, see [Full Identity Map](http://wiki.eclipse.org/Introduction_to_Cache_%28ELUG%29#Full_Identity_Map). * Weak – This option is similar to Full, except that objects are referenced using weak references. This option uses less memory than Full, but does not provide a durable caching strategy across client/server transactions. We recommend using this identity map for transactions that, once started, stay on the server side. For more information, see [Weak Identity Map](http://wiki.eclipse.org/Introduction_to_Cache_%28ELUG%29#Weak_Identity_Map). * 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. For more information, see [Soft Identity Map](http://wiki.eclipse.org/Introduction_to_Cache_%28ELUG%29#Soft_Identity_Map). * SoftWeak – This option is similar to Weak except that it maintains a most frequently used subcache that uses soft references. We recommend using this identity map in most circumstances as a means to control memory used by the cache. For more information, see [Soft Cache Weak Identity Map and Hard Cache Weak Identity Map](http://wiki.eclipse.org/Introduction_to_Cache_%28ELUG%29#Soft_Cache_Weak_Identity_Map_and_Hard_Cache_Weak_Identity_Map). * HardWeak – This option is similar to Weak except that it maintains a most frequently used subcache that uses hard references. For more information, see [Soft Cache Weak Identity Map and Hard Cache Weak Identity Map](http://wiki.eclipse.org/Introduction_to_Cache_%28ELUG%29#Soft_Cache_Weak_Identity_Map_and_Hard_Cache_Weak_Identity_Map). * NONE – This option does not preserve object identity and does not cache objects. Oracle does not recommend using this option. For more information, see [No Identity Map](http://wiki.eclipse.org/Introduction_to_Cache_%28ELUG%29#No_Identity_Map) or to turn off the cache - use**"eclipselink.cache.shared"="false"**.   **Note**: The values are case-sensitive.  **Note**: Using this property, you can override the @Cache annotation (see [How to Use the @Cache Annotation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Cache_Annotation)) attribute [type](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#type).  **Example**: persistence.xml file  <property name="eclipselink.cache.type.default" value="Full"/>  **Example**: property Map  import org.eclipse.persistence.config.CacheType;  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.CACHE\_TYPE\_DEFAULT, CacheType.Full); | SoftWeak |
| eclipselink.cache.size.default | The default maximum number of objects allowed in an EclipseLink cache.  Valid values: 0 to Integer.MAX\_VALUE as a String.  **Example**: persistence.xml file  <property name="eclipselink.cache.size.default" value="5000"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.CACHE\_SIZE\_DEFAULT, 5000); | 100 |
| eclipselink.cache.shared.default | The default for whether or not the EclipseLink session cache is shared by multiple client sessions.  The following are the valid values:   * true – The session cache services all clients attached to the session. When you read objects from or write objects to the data source using a client session, EclipseLink saves a copy of the objects in the parent server session's cache and makes them accessible to all other processes in the session. * false – The session cache services a single, isolated client exclusively. The isolated client can reference objects in a shared session cache but no client can reference objects in the isolated client's exclusive cache.   **Example**: persistence.xml file  <property name="eclipselink.cache.shared.default" value="false"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.CACHE\_SHARED\_DEFAULT, "false"); | true |
| eclipselink.cache.type.<ENTITY> | The type of session cache for the JPA entity named <ENTITY> or with the class name <ENTITY>. For more information on entity names, see Section 8.1 "Entity" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220).  The following are the valid values for the use in a persistence.xml file and for theorg.eclipse.persistence.config.CacheType:   * "Full" – see [eclipselink.cache.type.default](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.cache.type.default) * "HardWeak" – see [eclipselink.cache.type.default](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.cache.type.default) * "NONE" – see [eclipselink.cache.type.default](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.cache.type.default) * "SoftWeak" – see [eclipselink.cache.type.default](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.cache.type.default) * "Weak" – see [eclipselink.cache.type.default](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.cache.type.default) Note: Using this property, you can override the [@Cache](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Cache_Annotation) annotation's attribute [type](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#type).   **Example**: persistence.xml file  <property name="eclipselink.cache.type.Order" value="Full"/>  **Example**: property Map  import org.eclipse.persistence.config.CacheType;  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.CACHE\_TYPE+".Order", CacheType.Full); | eclipselink.cache.type.default |
| eclipselink.cache.size.<ENTITY> | The maximum number of JPA entities of the type denoted by JPA entity name <ENTITY> allowed in an EclipseLink cache. For more information on entity names, see Section 8.1 "Entity" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220).  Valid values: 0 to Integer.MAX\_VALUE as a String.  **Example**: persistence.xml file  <property name="eclipselink.cache.size.Order" value="5000"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.CACHE\_SIZE+".Order", 1000); | eclipselink.cache.size.default |
| eclipselink.cache.shared.<ENTITY> | Whether or not the EclipseLink session cache is shared by multiple client sessions for JPA entities of the type denoted by JPA entity name <ENTITY>. For more information on entity names, see Section 8.1 "Entity" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220).  The following are the valid values:   * true – The session cache services all clients attached to the session. When you read objects from or write objects to the data source using a client session, EclipseLink saves a copy of the objects in the parent server session's cache and makes them accessible to all other processes in the session. * false – The session cache services a single, isolated client exclusively. The isolated client can reference objects in a shared session cache but no client can reference objects in the isolated client's exclusive cache.   **Example**: persistence.xml file  <property name="eclipselink.cache.shared.Order" value="true"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.CACHE\_SHARED+".Order", "true"); | eclipselink.cache.shared.default |
| eclipselink.flush-clear.cache | Defines the EntityManager cache behaviour after a call to the flush method followed by a call to the clear method. You can specify this property while creating either an EntityManagerFactory(either in the map passed to the createEntityManagerFactory method, or in the persistence.xmlfile), or an EntityManager (in the map passed to the createEntityManager method). Note that the latter overrides the former.  The following are the valid values for the use in a persistence.xml file and for theorg.eclipse.persistence.config.FlushClearCache:   * Drop – The call to the clear method results in a drop of the entire EntityManager’s cache. This mode is the fastest and uses the least memory. However, after commit the shared cache might potentially contain stale data. * DropInvalidate – Even though the call to the clear method results in a drop of the entireEntityManager’s cache, classes that have at least one object updated or deleted are invalidated in the shared cache at commit time. This mode is slower than Drop, but as efficient memory usage-wise, and prevents stale data. * Merge – The call to the clear method results in a drop from the EntityManager’s cache of objects that have not been flushed. This mode leaves the shared cache in a perfect state after commit. However, it is the least memory-efficient mode; the memory might even run out in a very large transaction.   **Example**: persistence.xml file  <property name="eclipselink.flush-clear.cache" value="Drop"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.FLUSH\_CLEAR\_CACHE, FlushClearCache.Drop); | DropInvalidate |

**How to Use the @TimeOfDay Annotation**

You can use the @TimeOfDay annotation to specify a time of day using a Calendar instance. By doing so, you configure cache expiry on an entity class.

@Target({})

@Retention(RUNTIME)

public @interface TimeOfDay {

int hour() default 0;

int minute() default 0;

int second() default 0;

int millisecond() default 0;

}

This table lists attributes of the @TimeOfDay annotation.

***Attributes of the @TimeOfDay Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| hour | Set this attribute to the int value representing an hour of the day. | 0 | optional |
| minute | Set this attribute to the int value representing a minute of the day. | 0 | optional |
| second | Set this attribute to the int value representing a second of the day. | 0 | optional |
| millisecond | Set this attribute to the int value representing a millisecond of the day. | 0 | optional |

**How to Use the @ExistenceChecking Annotation**

Use the @ExistenceChecking annotation to specify the type of checking that EclipseLink should use when determining if an Entity is new or already existing.

On a merge operation, this annotation determines whether or not EclipseLink should only use the cache to check if an object exists, or should read the object (from the database or cache).

@Target({TYPE})

@Retention(RUNTIME)

public @interface ExistenceChecking {

ExistenceType value() default CHECK\_CACHE;

}

You may define the @ExistenceChecking annotation on the following:

* @Entity (see Section 8.1 "Entity" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220));
* [@MappedSuperclass](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40MappedSuperclass);

This table lists attributes of the @ExistenceChecking annotation.

***Attributes of the @ExistenceChecking Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| value | Set this attribute to the type (org.eclipse.persistence.annotations.ExistenceTypeenumerated type) of the existence checking that you will be using to determine if an insert or an update operation should occur for an object.  The following are the valid values for the ExistenceType:   * CHECK\_CACHE – This option assumes that if the object's primary key does not includenull and it is in the cache, then this object must exist. * CHECK\_DATABASE – This option triggers the object existence check on a database. * ASSUME\_EXISTENCE – This option assumes that if the object's primary key does not include null, then the object must exist. You may choose this option if your application guarantees the existence checking, or is not concerned about it. * ASSUME\_NON\_EXISTENCE – This option assumes that the object does not exist. You may choose this option if your application guarantees the existence checking, or is not concerned about it. If you specify this option, EclipseLink will force the call of an insert operation. | One of the following default values applies:   * ExistenceType.CHECK\_CACHE - If either an@ExistenceChecking annotation or an XML element exists, but the value attribute is not set. * ExistenceType.CHECK\_DATABASE - If neither the@ExistenceChecking annotation nor an XML element exists. | optional |

**Using EclipseLink JPA Extensions for Customization and Optimization**

EclipseLink defines one descriptor customizer annotation – @Customizer (see [How to Use the @Customizer Annotation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40Customizer_Annotation)).

EclipseLink also provides a number of persistence unit properties that you can specify to configure EclipseLink customization and validation (see [How to Use the Persistence Unit Properties for Customization and Validation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_Persistence_Unit_Properties_for_Customization_and_Validation)). These properties may compliment or provide an alternative to the usage of annotations.

|  |
| --- |
| **Note:** Persistence unit properties always override the corresponding annotations' attributes.  For more information, see the following:   * [What You May Need to Know About Overriding Annotations in JPA](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Overriding_Annotations_in_JPA) * [What You May Need to Know About Using EclipseLink JPA Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Using_EclipseLink_JPA_Persistence_Unit_Properties) |

In addition, EclipseLink provides a persistence unit property that you can specify to optimize your application (see [How to Use the Persistence Unit Properties for Optimization](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_Persistence_Unit_Properties_for_Optimization)).

**How to Use the @Customizer Annotation**

Use the @Customizer annotation to specify a class that implements the org.eclipse.persistence.config.DescriptorCustomizer interface and that is to be run against a class' descriptor after all metadata processing has been completed. See [eclipselink.descriptor.customizer.<ENTITY>](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.descriptor.customizer..3CENTITY.3E) for more information.

@Target({TYPE})

@Retention(RUNTIME)

public @interface Customizer {

Class value();

}

You can define the @Customizer annotation on the following:

* @Entity (see Section 8.1 "Entity" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220));
* [@MappedSuperclass](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40MappedSuperclass)
* [@Embeddable](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Embeddable)

|  |
| --- |
| **Note:** A @Customizer is not inherited from its parent classes. |

This table lists attributes of the @Customizer annotation.

***Attributes of the @Customizer Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| value | Set this attribute to the Class of the descriptor customizer that you want to apply to your entity's descriptor. | no default | required |

This example shows how to use the @Customizer annotation.

***Usage of the @Customizer Annotation***

@Entity

@Table(name="EMPLOYEE")

@Customizer(mypackage.MyCustomizer.**class**)

**public** **class** Employee **implements** Serializable {

...

}

**How to Use the Persistence Unit Properties for Customization and Validation**

This table lists the persistence unit properties that you can define in a persistence.xml file to configure EclipseLink customization and validation.

***EclipseLink JPA Properties for Customization and Validation***

|  |  |  |
| --- | --- | --- |
| **Property** | **Usage** | **Default** |
| eclipselink.orm.throw.exceptions | Specify whether or not EclipseLink JPA should 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.  The following are the valid values:   * true – throw exceptions. * false – log warning only.   **Example**: persistence.xml file  <property name="oracle.orm.throw.exceptions" value="false"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.ECLIPSELINK\_ORM\_THROW\_EXCEPTIONS, "false"); | true |
| eclipselink.exception-handler | Specify an EclipseLink exception handler class: a Java class that implements theorg.eclipse.persistence.exceptions.ExceptionHandler interface and provides a default (zero-argument) constructor. Use this class’ handleException method, which takes a java.lang.RuntimeException, to rethrow the exception, throw a different exception, or retry a query or a database operation.  For more information, see [Exception Handlers](http://wiki.eclipse.org/Introduction_to_EclipseLink_Sessions_%28ELUG%29#Exception_Handlers).  Valid values: class name of an ExceptionHandler class fully qualified by its package name.  **Example**: persistence.xml file  <property name="eclipselink.exception-handler" value="my.package.MyExceptionHandler">  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.EXCEPTION\_HANDLER\_CLASS, "my.package.MyExceptionHandler"); |  |
| eclipselink.weaving | Control whether or not the weaving of the entity classes is performed. The EclipseLink JPA persistence provider uses weaving to enhance JPA entities for such things as lazy loading, change tracking, fetch groups, and internal optimizations.  The following are the valid values:   * true – weave entity classes dynamically. * false – do not weave entity classes. * static – weave entity classes statically.   This assumes that classes have already been statically woven. Run the static weaver on the classes before deploying them.  For more information, see the following:   * [Using EclipseLink JPA Weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_EclipseLink_JPA_Weaving) * [How to Configure Dynamic Weaving for JPA Entities Using the EclipseLink Agent](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Dynamic_Weaving_for_JPA_Entities_Using_the_EclipseLink_Agent) * [How to Configure Static Weaving for JPA Entities](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Static_Weaving_for_JPA_Entities)   **Example**: persistence.xml file  <property name="eclipselink.weaving" value="false"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.WEAVING, "false"); | true |
| eclipselink.weaving.lazy | Enable or disable the lazy one-to-one (see [@OneToOne](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40OneToOne)) and many-to-one (see [@ManyToOne](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40ManyToOne)) mapping through weaving.  The following are the valid values:   * true – enable lazy one-to-one/many-to-one mapping through weaving (default). * false – disable lazy one-to-one/many-to-one mapping through weaving.   **Note**: you may set this option only if the [eclipselink.weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.weaving) option is set to true or static. The purpose of theeclipselink.weaving.lazy option is to provide more granular control over weaving.  For more information, see the following:   * [Using EclipseLink JPA Weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_EclipseLink_JPA_Weaving) * [What You May Need to Know About EclipseLink JPA Lazy Loading](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_EclipseLink_JPA_Lazy_Loading)   **Example**: persistence.xml file  <property name="eclipselink.weaving.lazy" value="false"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.WEAVING\_LAZY, "false"); | true |
| eclipselink.weaving.changetracking | Enable or disable the AttributeLevelChangeTracking through weaving.  The following are the valid values:   * true – enable the AttributeLevelChangeTracking through weaving (default). When enabled, only classes with all mappings allowing change tracking have change tracking enabled. * false – disable the AttributeLevelChangeTracking through weaving. Use this setting if the following applies:   + you cannot weave at all;   + you do not want your classes to be changed during weaving (for example, for debugging purposes);   + you wish to disable this feature for configurations that do not support it (for example, you are mutating the java.util.Dateor java.util.Calendar, you are using property access but modifying the underlying instance variables).   **Note**: you may set this option only if the [eclipselink.weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.weaving) option is set to true or static. The purpose of theeclipselink.weaving.changetracking option is to provide more granular control over weaving.  For more information, see the following:   * [Using EclipseLink JPA Weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_EclipseLink_JPA_Weaving) * [Configuring Change Policy](http://wiki.eclipse.org/Configuring_a_Descriptor_%28ELUG%29#Configuring_Change_Policy)   **Example**: persistence.xml file  <property name="eclipselink.weaving.changetracking" value="false"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.WEAVING\_CHANGE\_TRACKING, "false"); | true |
| eclipselink.weaving.fetchgroups | Enable or disable fetch groups through weaving.  The following are the valid values:   * true – enable the use of fetch groups through weaving (default). * false – disable the use of fetch groups. Use this setting if the following applies:   + you cannot weave at all;   + you do not want your classes to be changed during weaving (for example, for debugging purposes);- you wish to disable this feature for configurations that do not support it.   **Note**: you may set this option only if the [eclipselink.weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.weaving) option is set to true or static. The purpose of theeclipselink.weaving.fetchgroups option is to provide more granular control over weaving.  For more information, see the following:   * [Using EclipseLink JPA Weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_EclipseLink_JPA_Weaving) * [Configuring Fetch Groups](http://wiki.eclipse.org/Configuring_a_Descriptor_%28ELUG%29#Configuring_Fetch_Groups)   **Example**: persistence.xml file  <property name="eclipselink.weaving.fetchgroups" value="false"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.WEAVING\_FETCHGROUPS, "false"); | true |
| eclipselink.weaving.internal | Enable or disable internal optimizations through weaving.  The following are the valid values:   * true – enable internal optimizations through weaving (default). * false – disable internal optimizations through weaving.   **Note**: you may set this option only if the [eclipselink.weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.weaving) option is set to true or static. The purpose of theeclipselink.weaving.internal option is to provide more granular control over weaving.  For more information, see [Using EclipseLink JPA Weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_EclipseLink_JPA_Weaving).  **Example**: persistence.xml file  <property name="eclipselink.weaving.internal" value="false"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.WEAVING\_INTERNAL, "false"); | true |
| eclipselink.weaving.eager | Enable or disable indirection on eager relationships through weaving. (Even one-to-one and many-to-one relations with FetchType.EAGER become effectively "lazy".)  The following are the valid values:   * true – enable indirection on eager relationships through weaving. * false – disable indirection on eager relationships through weaving (default).   **Note**: you may set this option only if the [eclipselink.weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.weaving) option is set to true or static. The purpose of theeclipselink.weaving.eager option is to provide more granular control over weaving.  For more information, see the following:   * [Using EclipseLink JPA Weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_EclipseLink_JPA_Weaving) * [Value Holder Indirection](http://wiki.eclipse.org/Introduction_to_Mappings_%28ELUG%29#Value_Holder_Indirection)   **Example**: persistence.xml file  <property name="eclipselink.weaving.eager" value="true"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.WEAVING\_EAGER, "true"); | false |
| eclipselink.session.customizer | Specify an EclipseLink session customizer class: a Java class that implements theorg.eclipse.persistence.config.SessionCustomizer interface and provides a default (zero-argument) constructor. Use this class'customize method, which takes an org.eclipse.persistence.sessions.Session, to programmatically access advanced EclipseLink session API.  For more information, see [Session Customization](http://wiki.eclipse.org/Introduction_to_EclipseLink_Sessions_%28ELUG%29#Session_Customization).  Valid values: class name of a SessionCustomizer class fully qualified by its package name.  **Example**: persistence.xml file  <property name="eclipselink.session.customizer" value="acme.sessions.MySessionCustomizer"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.SESSION\_CUSTOMIZER, "acme.sessions.MySessionCustomizer"); |  |
| eclipselink.descriptor.customizer.<ENTITY> | Specify an EclipseLink descriptor customizer class – a Java class that implements theorg.eclipse.persistence.config.DescriptorCustomizer interface and provides a default (zero-argument) constructor. Use this class's customize method, which takes an org.eclipse.persistence.descriptors.ClassDescriptor, to programmatically access advanced EclipseLink descriptor and mapping API for the descriptor associated with the JPA entity named <ENTITY>.  For more information on entity names, see Section 8.1 "Entity" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220)  For more information, see [Descriptor Customization](http://wiki.eclipse.org/Introduction_to_Descriptors_%28ELUG%29#Descriptor_Customization).  **Note**: EclipseLink does not support multiple descriptor customizers.Valid values: class name of a DescriptorCustomizer class fully qualified by its package name.  **Example**: persistence.xml file  <property name="eclipselink.descriptor.customizer.Order" value="acme.sessions.MyDescriptorCustomizer"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.DESCRIPTOR\_CUSTOMIZER+".Order", "acme.sessions.MyDescriptorCustomizer"); |  |
| eclipselink.validation-only | Specify whether or not deployment should be for validation only.  The following are the valid values:   * true – deployment is only for validation; the EclipseLink session will not log in. * false – normal deployment; the EclipseLink session will log in.   **Example**: persistence.xml file  <property name="eclipselink.validation-only" value="false"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.ECLIPSELINK\_VALIDATION\_ONLY\_PROPERTY, "false"); | true |
| eclipselink.classloader | Specify the class loader to use for creation of an EntityManagerFactory in the property map passed to thecreateEntityManagerFactory method.  **Example**: persistence.xml file  <property name="eclipselink.classloader" value="MyClassloader"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.CLASSLOADER, "MyClassloader"); |  |

For information about the use of annotations as opposed to persistence unit properties and vice versa, see the following:

* [What You May Need to Know About Overriding Annotations in JPA](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Overriding_Annotations_in_JPA)
* [What You May Need to Know About EclipseLink JPA Overriding Mechanisms](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_EclipseLink_JPA_Overriding_Mechanisms)

For more information about persistence unit properties, see [What You May Need to Know About Using EclipseLink JPA Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Using_EclipseLink_JPA_Persistence_Unit_Properties).

**How to Use the Persistence Unit Properties for Optimization**

This table lists the persistence unit properties that you can define in a persistence.xml file to optimize your EclipseLink application.

***EclipseLink JPA Persistence Unit Properties for Optimization***

|  |  |  |
| --- | --- | --- |
| **Property** | **Usage** | **Default** |
| eclipselink.profiler | The type of the performance profiler.  For more information on performance profilers, see [Optimizing the EclipseLink Application](http://wiki.eclipse.org/Optimizing_the_EclipseLink_Application_%28ELUG%29).  The following are the valid values for the use in a persistence.xml file and for theorg.eclipse.persistence.config.ProfilerType:   * PerformanceProfiler – Use EclipseLink performance profiler (org.eclipse.persistence.tools.profiler.PerformanceProfiler class). For more information, see [Measuring EclipseLink Performance with the EclipseLink Profiler](http://wiki.eclipse.org/Optimizing_the_EclipseLink_Application_%28ELUG%29#Measuring_EclipseLink_Performance_with_the_EclipseLink_Profiler). * QueryMonitor – Monitor query executions and cache hits (org.eclipse.persistence.tools.profiler.QueryMonitorclass).This option provides a simple low-overhead means for measuring performance of query executions and cache hits. You may want to use this option for performance analysis in a complex system. * NoProfiler – Do not use a performance profiler. * Custom profiler – Use your own custom profiler class. Create it by implementing theorg.eclipse.persistence.sessions.SessionProfiler interface and providing a no-argument constructor.   **Example**: persistence.xml file  <property name="eclipselink.profiler" value="PerformanceProfiler"/>  **Example**: property Map  import org.eclipse.persistence.config.ProfilerType;  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.PROFILER, ProfilerType.PerformanceProfiler);  **Example**: persistence.xml file  <property name="eclipselink.profiler" value="mypackage.com.MyProfiler"/>  **Note**: Ensure that MyProfiler is in your classpath.  **Example**: property Map  import org.eclipse.persistence.config.ProfilerType;  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.PROFILER, "mypackage.com.MyProfiler"); | NoProfiler |
| eclipselink.persistence.context.reference-mode | Specify whether or not the Java VM is allowed to manage the number of objects within the persistence context using Java’s weak references.  In cases where your application cannot use the EntityManager clear method to clear an extended persistence context, use this setting to enable the VM to remove unreferenced and unchanged objects from the persistence context, resulting in making resources available for other uses.  You can set this property either during the EntityManagerFactory createEntityManager(Map) call, or globally in thepersistence.xml file.  The following are the valid values for the use in a persistence.xml file and for theorg.eclipse.persistence.sessions.factories.ReferenceMode:   * HARD - Use this option to specify that all references to all objects will be through hard references. These objects will not be available for garbage collection until the referencing artifact (such as persistence context or unit of work) is released/cleared or closed. * WEAK - Use this option to specify that references to objects supporting active attribute change tracking (see [Attribute Change Tracking Policy](http://wiki.eclipse.org/Introduction_to_EclipseLink_Transactions_%28ELUG%29#Attribute_Change_Tracking_Policy)) will be held by weak references. That is, any object no longer referenced directly or indirectly will be available for garbage collection. When a change is made to a change-tracked object, that object is moved to a hard reference and will not be available for garbage collection until flushed. New and removed objects, as well as objects that do not support active attribute change tracking, will also be held by hard references and will not be available for garbage collection. * FORCE\_WEAK - Use this option to specify that all objects, including non-change-tracked objects, are to be held by weak references. When a change is made to a change-tracked object, that object is moved to a hard reference and will not be available for garbage collection until flushed. However, any objects that do not support active attribute change tracking may be garbage collected before their changes are flushed to a database, which can potentially result in a loss of changes. New and removed objects will be held by hard references and will not be available for garbage collection.   Using Java, you can configure reference mode through the EclipseLink Session's acquireUnitOfWork(ReferenceMode mode) method. To specify a persistence unit- or session-wide default, use the Session's setDefaultReferenceMode method.  For more information, see the following:   * [Using EclipseLink JPA Extensions for Tracking Changes](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_EclipseLink_JPA_Extensions_for_Tracking_Changes) * [Cache Type and Object Identity](http://wiki.eclipse.org/Introduction_to_Cache_%28ELUG%29#Cache_Type_and_Object_Identity)   **Example**: persistence.xml file  <property name="eclipselink.persistence-context.reference-mode" value="FORCE\_WEAK"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.PERSISTENCE\_CONTEXT\_REFERENCE\_MODE, ReferenceMode.FORCE\_WEAK); | WEAK |

For information about optimization, see [Optimizing the EclipseLink Application](http://wiki.eclipse.org/Optimizing_the_EclipseLink_Application_%28ELUG%29)

For more information about persistence unit properties, see [What You May Need to Know About Using EclipseLink JPA Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Using_EclipseLink_JPA_Persistence_Unit_Properties).

**Using EclipseLink JPA Extensions for Copy Policy**

The copy policy enables EclipseLink to produce exact copies of persistent objects. For more information, see [Configuring Copy Policy](http://wiki.eclipse.org/Configuring_a_Descriptor_%28ELUG%29#Configuring_Copy_Policy).

EclipseLink defines the following copy policy annotations:

* [@CopyPolicy](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40CopyPolicy_Annotation)
* [@CloneCopyPolicy](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40CloneCopyPolicy_Annotation)
* [@InstantitationCopyPolicy](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40InstantitationCopyPolicy_Annotation)

**How to Use the @CopyPolicy Annotation**

Use the @CopyPolicy annotation to specify a class that implements the org.eclipse.persistence.descriptors.copying.CopyPolicy interface to set the copy policy on anEntity.

@Target({TYPE})

@Retention(RUNTIME)

public @interface CopyPolicy {

Class value();

}

You can define the @CopyPolicy annotation on the following:

* @Entity (see Section 8.1 "Entity" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220));
* [@MappedSuperclass](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40MappedSuperclass)

This table lists attributes of the @CopyPolicy annotation.

***Attributes of the @CopyPolicy Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| value | Set this attribute to the Class of the copy policy that you want to apply to your entity's descriptor. The class must implement org.eclipse.persistence.descriptors.copying.CopyPolicy | no default | required |

This example shows how to use the @CopyPolicy annotation.

***Usage of the @CopyPolicy Annotation***

@Entity

@Table(name="EMPLOYEE")

@CopyPolicy(mypackage.MyCopyPolicy.**class**)

**public** **class** Employee **implements** Serializable {

...

}

**How to Use the @CloneCopyPolicy Annotation**

Use the @CloneCopyPolicy annotation to set the clone copy policy (org.eclipse.persistence.descriptors.copying.CloneCopyPolicy) on an Entity.

@Target({TYPE})

@Retention(RUNTIME)

public @interface CloneCopyPolicy {

Sting method();

Sting workingCopyMethod();

}

The @CloneCopyPolicy must specify one or both of the method or workingCopyMethod attributes based on the following:

* Use the method for the clone whose function is comparison in conjunction with EclipseLink's DeferredChangeDetectionPolicy (see [Deferred Change Detection Policy](http://wiki.eclipse.org/Introduction_to_EclipseLink_Transactions_%28ELUG%29#Deferred_Change_Detection_Policy)).
* Use the workingCopyMethod to clone objects that will be returned, as they are registered in EclipseLink's transactional mechanism (the unit of work).

You can define the @CloneCopyPolicy annotation on the following:

* @Entity (see Section 8.1 "Entity" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220));
* [@MappedSuperclass](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40MappedSuperclass)

This table lists attributes of the @CloneCopyPolicy annotation.

***Attributes of the @CloneCopyPolicy Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| method | Set this attribute to the String method name that EclipseLink will use to create a clone to enable comparison by EclipseLink's[deferred change detection policy](http://wiki.eclipse.org/Introduction_to_EclipseLink_Transactions_%28ELUG%29#Deferred_Change_Detection_Policy)) that you want to apply to your entity's descriptor.  Note: you have to set either this attribute, or the workingCopyMethod, or both. | no default | optional/required |
| workingCopyMethod | Set this attribute to the String method name that EclipseLink will use to create the object returned when registering an Objectin an EclipseLink [unit of work](http://wiki.eclipse.org/Introduction_to_EclipseLink_Transactions_%28ELUG%29).  Note: you have to set either this attribute, or the method, or both. | no default | optional/required |

**How to Use the @InstantiationCopyPolicy Annotation**

Instantiation copy policy is the default copy policy in EclipseLink. Use the @InstantiationCopyPolicy annotation to override other types of copy policies for an Entity.

@Target({TYPE})

@Retention(RUNTIME)

public @interface CloneCopyPolicy {

}

You can define the @InstantiationCopyPolicy annotation on the following:

* @Entity (see Section 8.1 "Entity" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220));
* [@MappedSuperclass](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40MappedSuperclass)

The @InstantiationCopyPolicy annotation does not have attributes.

**Using EclipseLink JPA Extensions for Declaration of Read-Only Classes**

EclipseLink defines one annotation that you can use to declare classes as read-only – [@ReadOnly](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40ReadOnly_Annotation).

For more information, see the following:

* [Configuring Read-Only Descriptors](http://wiki.eclipse.org/Configuring_a_Descriptor_%28ELUG%29#Configuring_Read-Only_Descriptors)
* [Declaring Read-Only Classes](http://wiki.eclipse.org/Using_Advanced_Unit_of_Work_API_%28ELUG%29#Declaring_Read-Only_Classes)

**How to Use the @ReadOnly Annotation**

Use the @ReadOnly annotation to specify that a class is read-only (see [Declaring Read-Only Classes](http://wiki.eclipse.org/Using_Advanced_Unit_of_Work_API_%28ELUG%29#Declaring_Read-Only_Classes)).

|  |
| --- |
| **Note:** Any changes made within a managed instance in a transaction or to a detached instance and merged will have no effect in the context of a read-only entity class. |

@Target({TYPE})

@Retention(RUNTIME)

public @interface ReadOnly {}

You can define the @ReadOnly annotation on the following:

* @Entity (see Section 8.1 "Entity" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220));
* [@MappedSuperclass](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40MappedSuperclass);
* the root of the inheritance hierarchy (if applicable).

The @ReadOnly annotation does not have attributes.

This example shows how to use the @ReadOnly annotation.

***Usage of the @ReadOnly Annotation***

@Entity

@ReadOnly

**public** **class** Employee **implements** Serializable {

...

}

**Using EclipseLink JPA Extensions for Returning Policy**

The returning policy enables INSERT or UPDATE operations to return values back into the object being written. These values include table default values, trigger or stored procedures computed values. For more information, see [Configuring Returning Policy](http://wiki.eclipse.org/Configuring_a_Descriptor_%28ELUG%29#Configuring_Returning_Policy).

EclipseLink defines the following returning policy annotations:

* [@ReturnInsert](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40ReturnInsert_Annotation)
* [@ReturnUpdate](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40ReturnUpdate_Annotation)

**How to Use the @ReturnInsert Annotation**

You can only specify the @ReturnInsert for a [@Basic](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Basic) mapping.

@Target({METHOD, FIELD})

@Retention(RUNTIME)

public @interface ReturnInsert {

boolean returnOnly() default false;

}

This table lists attributes of the @ReturnInsert annotation.

***Attributes of the @ReturnInsert Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| returnOnly | Set this attribute to the boolean value of true if you want a return of a value for this field, without including the field in the insert. | false | optional |

The [Usage of the @ReturnInsert Annotation Without Arguments](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-15) example shows how to use the @ReturnInsert annotation without specifying the value for the returnOnlyargument, therefore accepting the default value of false. The [Usage of the @ReturnInsert Annotation with Arguments](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-16) example shows how to set the value of the returnOnlyargument to true.

***Usage of the @ReturnInsert Annotation Without Arguments***

@ReturnInsert

**public** String getFirstName() {

**return** firstName;

}

***Usage of the @ReturnInsert Annotation with Arguments***

@ReturnInsert(returnOnly=**true**)

**public** String getFirstName() {

**return** firstName;

}

**How to Use the @ReturnUpdate Annotation**

You can only specify the @ReturnUpdate for a [@Basic](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Basic) mapping.

@Target({METHOD, FIELD})

@Retention(RUNTIME)

public @interface ReturnUpdate {}

The @ReturnUpdate annotation does not have attributes.

This example shows how to use the @ReturnUpdate annotation.

***Usage of the @ReturnUpdate Annotation***

@ReturnUpdate

public String getFirstName() {

return firstName;

}

**Using EclipseLink JPA Extensions for Optimistic Locking**

EclipseLink defines one annotation for optimistic locking – [@OptimisticLocking](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40OptimisticLocking_Annotation).

For more information, see the following:

* [Optimistic Locking](http://wiki.eclipse.org/Introduction_to_EclipseLink_Application_Development_%28ELUG%29#Optimistic_Locking)
* [Configuring an Optimistic Locking Policy](http://wiki.eclipse.org/Configuring_a_Descriptor_%28ELUG%29#Configuring_an_Optimistic_Locking_Policy)

**How to Use the @OptimisticLocking Annotation**

You can use the @OptimisticLocking annotation to specify the type of optimistic locking that EclipseLink should use when updating or deleting entities.

|  |
| --- |
| **Note:** EclipseLink supports additional optimistic locking policies beyond what is supported through the JPA specification (such as@Version - see Section 9.1.17 "Version Annotation" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220)). When mapping to a database schema where a version column does not exist and cannot be added, these locking policies enable the concurrency protection. |

@Target({TYPE})

@Retention(RUNTIME)

public @interface OptimisticLocking {

OptimisticLockingType type() default VERSION\_COLUMN;

Column[] selectedColumns() default {};

boolean cascade() default false;

}

This table lists attributes of the @OptimisticLocking annotation.

***Attributes of the @OptimisticLocking Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| type | Set this attribute to the type (org.eclipse.persistence.annotations.OptimisticLockingTypeenumerated type) of the optimistic locking policy that you will be using.  The following are the valid values for the OptimisticLockingType:   * ALL\_COLUMNS – Use this type of locking policy to compare every field in the table in the WHEREclause during an update or a delete operation. If any field has been changed, EclipseLink will throw an optimistic locking exception. * CHANGED\_COLUMNS – Use this type of locking policy to compare changed fields in the table in theWHERE clause during an update operation. If any field has been changed, EclipseLink will throw an optimistic locking exception.  Note: performing the same during a delete operation will only compare primary keys. * SELECTED\_COLUMNS – Use this type of locking policy to compare selected fields in the table in theWHERE clause during an update or a delete operation. If any field has been changed, EclipseLink will throw an optimistic locking exception.  Note: specified fields must be mapped and must not be primary keys.  Note: EclipseLink will throw an exception if you set the SELECTED\_COLUMNS type, but fail to specify the selectedColumns. You must also specify the name attribute of the Column. * VERSION\_COLUMN – Use this type of locking policy to compare a single version number in theWHERE clause during an update operation.  Note: the version field must be mapped and must not be the primary key.  Note: this functionality is equivalent to the functionality of the @Version annotation (see Section 9.1.17 "Version Annotation" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220)) in JPA. If you use this option, you must also provide the @Version annotation on the version field or property. For more information, see [What You May Need to Know About Version Fields](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Version_Fields). | OptimisticLockingType.VERSION\_COLUMN | optional |
| selectedColumns | Set this attribute to an array of javax.persistence.Column instances.  For an optimistic locking policy of type [SELECTED\_COLUMNS](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#SELECTED_COLUMNS), this annotation member becomes a required field.  Note: EclipseLink will throw an exception if you set the [SELECTED\_COLUMNS](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#SELECTED_COLUMNS) type, but fail to specify the selectedColumns. You must also specify the name attribute of the Column. | empty Column array | optional |
| cascade | Set the value of this attribute to a boolean value of true to specify that the optimistic locking policy should cascade the lock.  By enabling cascading you configure EclipseLink to automatically force a version field update on a parent object when its privately owned child object's version field changes.  Note: In the current release, only supported with [VERSION\_COLUMN](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#VERSION_COLUMN) locking.  For more information, see the following:   * [Optimistic Version Locking Policies and Cascading](http://wiki.eclipse.org/Introduction_to_Descriptors_%28ELUG%29#Optimistic_Version_Locking_Policies_and_Cascading) * [Configuring Optimistic Locking Policy Cascading](http://wiki.eclipse.org/Configuring_a_Descriptor_%28ELUG%29#Configuring_Optimistic_Locking_Policy_Cascading) | false | optional |

|  |
| --- |
| **Note:** Setting an @OptimisticLocking may override any @Version specification (see Section 9.1.17 "Version Annotation" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220)) on the entity: EclipseLink will not throw an exception, but will log a warning.  You can specify @Version without any @OptimisticLocking specification to define a version locking policy (org.eclipse.persistence.descriptors.VersionLockingPolicy) on the source entity. |

This example shows how to use the @OptimisticLocking annotation with the ALL\_COLUMNS type.

***Usage of the @OptimisticLocking Annotation - ALL\_COLUMNS***

@Entity

@Table(name="EMPLOYEE")

@OptimisticLocking(type=OptimisticLockingType.ALL\_COLUMNS)

public class Employee implements Serializable{

private Integer id;

private String firstName;

private String lastName;

...

}

The following example shows how to use the @OptimisticLocking annotation with the CHANGED\_COLUMNS type.

***Usage of the @OptimisticLocking Annotation - CHANGED\_COLUMNS***

@Entity

@Table(name="EMPLOYEE")

@OptimisticLocking(type=OptimisticLockingType.CHANGED\_COLUMNS)

public class Employee implements Serializable{

private Integer id;

private String firstName;

private String lastName;

...

}

The following example shows how to use the @OptimisticLocking annotation with the SELECTED\_COLUMNS type.

***Usage of the @OptimisticLocking Annotation - SELECTED\_COLUMNS***

@Entity

@Table(name="EMPLOYEE")

@OptimisticLocking(

type=OptimisticLockingType.SELECTED\_COLUMNS,

selectedColumns={@Column(name="id"), @Column(name="lastName")}

)

public class Employee implements Serializable{

@Id

private Integer id;

private String lastName;

private String lastName;

...

}

The following example shows how to use the @OptimisticLocking annotation with the VERSION\_COLUMN type.

***Usage of the @OptimisticLocking Annotation - VERSION\_COLUMN***

@Entity

@Table(name="EMPLOYEE")

@OptimisticLocking(type=OptimisticLockingType.VERSION\_COLUMN, cascade=true)

public class Employee implements Serializable{

private String firstName;

private String lastName;

@Version private int version;

...

}

**Using EclipseLink JPA Extensions for Stored Procedure Query**

EclipseLink defines the following stored procedure query annotations:

* [@NamedStoredProcedureQuery](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40NamedStoredProcedureQuery_Annotation)
* [@StoredProcedureParameter](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40StoredProcedureParameter_Annotation)
* [@NamedStoredProcedureQueries](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40NamedStoredProcedureQueries_Annotation)

You can execute a stored procedure query like any other named query (see [Named Queries](http://wiki.eclipse.org/Introduction_to_EclipseLink_Queries_%28ELUG%29)). For more information, see [Queries](http://wiki.eclipse.org/EclipseLink/UserGuide/Queries_%28ELUG%29).

**How to Use the @NamedStoredProcedureQuery Annotation**

Use the @NamedStoredProcedureQuery to define queries that call stored procedures as named queries.

@Target({TYPE})

@Retention(RUNTIME)

public @interface NamedStoredProcedureQuery {

String name();

String procedureName();

QueryHint[] hints() default {};

Class resultClass() default void.class;

String resultSetMapping() default "";

boolean returnsResultSet() default false;

StoredProcedureParameter[] parameters() default {};

}

This table lists attributes of the @NamedStoredProcedureQuery annotation.

***Attributes of the @NamedStoredProcedureQuery Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| name | Set this attribute to the unique String name that references this stored procedure query. | no default | required |
| hints | Set this attribute to an array of javax.persistence.QueryHint instances. | empty QueryHint array | optional |
| resultClass | Set this attribute to the Class of the query result. | void.class | optional |
| resultSetMapping | Set this attribute to the String name of the javax.persistence.SQLResultSetMapping instance. | empty String | optional |
| procedureName | Set this attribute to the String name of the stored procedure. | no default | required |
| returnsResultSet | Set this attribute to the boolean value of false to disable the return of the result set. | true | optional |
| parameters | Set the value of this attribute to an array of [@StoredProcedureParameter](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_StoredProcedureParameter_Annotation) instances to define arguments to the stored procedure. | emptyStoredProcedureParameterarray | optional |

This example shows how to use the @NamedStoredProcedureQuery annotation.

***Usage of the @NamedStoredProcedureQuery Annotation***

@Entity

@Table(name="EMPLOYEE")

@NamedStoredProcedureQuery(

name="ReadEmployee",

procedureName="Read\_Employee",

parameters={

@StoredProcedureParameter(queryParamater="EMP\_ID")}

)

public class Employee implements Serializable{

...

}

**How to Use the @StoredProcedureParameter Annotation**

Use the @StoredProcedureParameter annotation within a [@NamedStoredProcedureQuery](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_NamedStoredProcedureQuery_Annotation) annotation to define arguments to the stored procedure.

@Target({})

@Retention(RUNTIME)

public @interface StoredProcedureParameter {

String queryParameter();

Direction direction() default IN;

int jdbcType() default -1;

String jdbcTypeName() default "";

String name() default "";

Class type() default void.class;

}

This table lists attributes of the @StoredProcedureParameter annotation.

***Attributes of the @StoredProcedureParameter Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| queryParameter | Set this attribute to the String query parameter name. | no default | required |
| direction | Set the value of this attribute to define the direction(org.eclipse.persistence.annotations.Direction enumerated type) of the stored procedure parameter.  The following are valid values for Direction.IN:   * IN – Input parameter. * OUT – Output parameter. * IN\_OUT – Input and output parameter. * OUT\_CURSOR – Output cursor. Note: EclipseLink will throw an exception if you set more than one parameter to the OUT\_CURSOR type. | emptyDirection.IN | optional |
| name | Set this attribute to the String name of the stored procedure parameter. | "" | optional |
| type | Set this attribute to the type of Java Class that you want to receive back from the procedure. This depends on the type returned from the procedure. | void.class | optional |
| jdbcType | Set this attribute to the int value of JDBC type code. This depends on the type returned from the procedure. | -1 | optional |
| jdbcTypeName | Set this attribute to the String value of the JDBC type name. Note: setting of this attribute may be required for ARRAY or STRUCT types. | "" | optional |

The [Usage of the @NamedStoredProcedureQuery Annotation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-25) example shows how to use the @StoredProcedureParameter annotation.

For more information, see the following:

* [StoredProcedureCall](http://wiki.eclipse.org/Introduction_to_EclipseLink_Queries_%28ELUG%29#StoredProcedureCall)
* [Using a StoredProcedureCall](http://wiki.eclipse.org/Using_Basic_Query_API_%28ELUG%29)
* [Call Queries](http://wiki.eclipse.org/Introduction_to_EclipseLink_Queries_%28ELUG%29#Call_Queries)

**How to Use the @NamedStoredProcedureQueries Annotation**

Use the @NamedStoredProcedureQueries to define queries that call stored procedures as named queries.

@Target({TYPE})

@Retention(RUNTIME)

public @interface NamedStoredProcedureQueries {

NamedStoredProcedureQuery[] value();

}

This table lists attributes of the @NamedStoredProcedureQueries annotation.

***Attributes of the @NamedStoredProcedureQueries Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| value | Set this attribute to the array of the [@NamedStoredProcedureQuery](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40NamedStoredProcedureQuery_Annotation) annotations. | no default | required |

**Using EclipseLink JPA Extensions for JDBC**

EclipseLink JPA provides persistence unit properties that you can define in a persistence.xml file to configure how EclipseLink will use the connections returned from the data source used. These properties are divided into the two following categories:

* Options that you can use to configure how EclipseLink communicates with the JDBC connection (see [How to Use EclipseLink JPA Extensions for JDBC Connection Communication](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_EclipseLink_JPA_Extensions_for_JDBC_Connection_Communication)).
* Options that you can use to configure EclipseLink own connection pooling (see [How to Use EclipseLink JPA Extensions for JDBC Connection Pooling](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_EclipseLink_JPA_Extensions_for_JDBC_Connection_Pooling)).

**How to Use EclipseLink JPA Extensions for JDBC Connection Communication**

This table lists the EclipseLink JPA persistence unit properties that you can define in a persistence.xml file to configure how EclipseLink communicates with the JDBC connection.

***EclipseLink JPA Persistence Unit Properties for JDBC Connection Communication***

|  |  |  |
| --- | --- | --- |
| **Property** | **Usage** | **Default** |
| eclipselink.jdbc.bind-parameters | Control whether or not the query uses parameter binding1.  For more information, see [How to Use Parameterized SQL (Parameter Binding) and Prepared Statement Caching for Optimization](http://wiki.eclipse.org/Optimizing_the_EclipseLink_Application_%28ELUG%29).  The following are the valid values:   * true – bind all parameters. * false – do not bind parameters.   **Example**: persistence.xml file  <property name="eclipselink.jdbc.bind-parameters" value="false"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.JDBC\_BIND\_PARAMETERS, "false"); | true |
| eclipselink.jdbc.native-sql | Enable or disable EclipseLink's generation of database platform-specific SQL (as opposed to generic SQL).2  The following are the valid values:   * true – enable EclipseLink's generation of database platform-specific SQL. * false – disable generation of database platform-specific SQL by EclipseLink.   **Example**: persistence.xml file  <property name="eclipselink.jdbc.native-sql" value="true"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.NATIVE\_SQL, "true"); | false |
| eclipselink.jdbc.batch-writing | Specify the use of batch writing to optimize transactions with multiple write operations2.  Set the value of this property into the session at deployment time.  The following are the valid values for the use in a persistence.xml file and for theorg.eclipse.persistence.config.BatchWriting:   * JDBC – use JDBC batch writing. * Buffered – do not use either JDBC batch writing nor native platform batch writing. * Oracle-JDBC – use both JDBC batch writing and Oracle native platform batch writing. Use OracleJDBC in your property map. * None – do not use batch writing (turn it off).   Note: if you set any other value, EclipseLink will throw an exception.  **Example**: persistence.xml file  <property name="eclipselink.jdbc.batch-writing" value="Oracle-JDBC"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.BATCH\_WRITING, BatchWriting.OracleJDBC); | None |
| eclipselink.jdbc.cache-statements | Enable or disable EclipseLink internal statement caching.2  Note: we recommend enabling this functionality if you are using EclipseLink connection pooling.  The following are the valid values:   * true – enable EclipseLink's internal statement caching. * false – disable internal statement caching.   **Example**: persistence.xml file  <property name="eclipselink.jdbc.cache-statements" value="false"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.CACHE\_STATEMENTS, "false"); | false |
| eclipselink.jdbc.cache-statements.size | The number of statements held when using internal statement caching.2  Set the value at the deployment time.  Valid values: 0 to Integer.MAX\_VALUE (depending on your JDBC driver) as a String.  **Example**: persistence.xml file  <property name="eclipselink.jdbc.cache-statements.size" value="2"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.CACHE\_STATEMENTS\_SIZE, "2"); | 50 |
| eclipselink.jdbc.exclusive-connection.is-lazy | Specify when a write connection is acquired lazily. For more information, see [Lazy Connection Acquisition](http://wiki.eclipse.org/Configuring_a_Session_%28ELUG%29#Lazy_Connection_Acquisition).  The following are the valid values:   * true - aquire the write connection lazily. * false - do not aquire the write connection lazily.   For more information, see [Configuring Connection Policy](http://wiki.eclipse.org/Configuring_a_Session_%28ELUG%29#Configuring_Connection_Policy).  **Example**: persistence.xml file  <property name="eclipselink.jdbc.exclusive-connection.is-lazy" value="false"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.EXCLUSIVE\_CONNECTION\_IS\_LAZY, "false"); | true |
| eclipselink.jdbc.exclusive-connection.mode | Specify when EclipseLink should perform reads through a write connection. For more information, see [Exclusive Write Connections](http://wiki.eclipse.org/Configuring_a_Session_%28ELUG%29#Exclusive_Write_Connections).  You can set this property while creating either an EntityManagerFactory (either in the map passed to thecreateEntityManagerFactory method, or in the persistence.xml file), or an EntityManager (in the map passed to thecreateEntityManager method). Note that the latter overrides the former.  The following are the valid values for the use in a persistence.xml file and for theorg.eclipse.persistence.config.ExclusiveConnectionMode:   * Transactional - Create an isolated client session (see [Isolated Client Sessions](http://wiki.eclipse.org/Introduction_to_EclipseLink_Sessions_%28ELUG%29#Isolated_Client_Sessions)) if some or all entities require isolated cache4; otherwise, create a client session. Note: EclipseLink keeps the connection exclusive for the duration of the transaction. Inside the transaction, EclipseLink performs all writes and reads through the exclusive connection. However, outside the Eclipelink transaction, a new connection is acquired from the connection pool for each read and released back immediately after the query is executed. * Isolated - Create an exclusive isolated client session if reading an isolated Entity; otherwise, raise an error. Note: EclipseLink keeps the connection exclusive for the lifetime of the owning EntityManager. Inside the transaction, EclipseLink performs all writes and reads through the exclusive connection. However, outside the Eclipelink transaction only isolated entities are read through the exclusive connection; for nonisolated entities, a new connection is acquired from the connection pool for each read and released back immediately after the query is executed. * Always - Create an exclusive isolated client session (see [Isolated Client Sessions](http://wiki.eclipse.org/Introduction_to_EclipseLink_Sessions_%28ELUG%29#Isolated_Client_Sessions)) if reading an isolated Entity; otherwise, create an exclusive client session. Note: EclipseLink keeps the connection exclusive for the lifetime of the owning EntityManager and performs all writes and reads through the exclusive connection.   For more information, see [Configuring Connection Policy](http://wiki.eclipse.org/Configuring_a_Session_%28ELUG%29#Configuring_Connection_Policy).  **Example**: persistence.xml file  <property name="eclipselink.jdbc.exclusive-connection.mode" value="Always"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.EXCLUSIVE\_CONNECTION\_MODE, "Always"); | Transactional |
| eclipselink.jdbc.driver | The class name of the JDBC driver you want to use, fully qualified by its package name. This class must be on your application classpath.3  **Example**: persistence.xml file  <property name="eclipselink.jdbc.driver" value="oracle.jdbc.driver.OracleDriver"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.JDBC\_DRIVER, "oracle.jdbc.driver.OracleDriver"); |  |
| eclipselink.jdbc.password | The password for your JDBC user.3  **Example**: persistence.xml file  <property name="eclipselink.jdbc.password" value="tiger"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.JDBC\_PASSWORD, "tiger"); |  |
| eclipselink.jdbc.url | The JDBC connection URL required by your JDBC driver.3  **Example**: persistence.xml file  <property name="eclipselink.jdbc.url" value="jdbc:oracle:thin:@MYHOST:1521:MYSID"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.JDBC\_URL, "jdbc:oracle:thin:@MYHOST:1521:MYSID"); |  |
| eclipselink.jdbc.user | The user name for your JDBC user.3  **Example**: persistence.xml file  <property name="eclipselink.jdbc.url" value="scott"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.JDBC\_USER, "scott"); |  |

1 This property applies when used in a Java SE environment.  
2 This property applies when used both in a Java SE and Java EE environment.  
3 This property applies when used in a Java SE environment or a resource-local persistence unit (see Section 5.5.2 "Resource-Local Entity Managers" and Section 6.2.1.2 "transaction-type" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220)).  
4 To do this, set the [eclipselink.cache.shared.<ENTITY>](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.cache.shared.ENTITY) property for one or more entities to false; Use the [eclipselink.cache.shared.default](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.cache.shared.default) property if you want to use the isolated cache for all entities.

For information about the use of annotations as opposed to persistence unit properties and vice versa, see the following:

* [What You May Need to Know About Overriding Annotations in JPA](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Overriding_Annotations_in_JPA)
* [What You May Need to Know About EclipseLink JPA Overriding Mechanisms](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_EclipseLink_JPA_Overriding_Mechanisms).

For more information about persistence unit properties, see [What You May Need to Know About Using EclipseLink JPA Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Using_EclipseLink_JPA_Persistence_Unit_Properties).

**How to Use EclipseLink JPA Extensions for JDBC Connection Pooling**

This table lists the EclipseLink JPA persistence unit properties that you can define in a persistence.xml file to configure EclipseLink [internal connection pooling](http://wiki.eclipse.org/Introduction_to_Data_Access_%28ELUG%29#Internal_Connection_Pools).

***EclipseLink JPA Persistence Unit Properties for JDBC Connection Pooling***

|  |  |  |
| --- | --- | --- |
| **Property** | **Usage** | **Default** |
| eclipselink.jdbc.read-connections.max | The maximum number of connections allowed in the JDBC read connection pool.1  Valid values: 0 to Integer.MAX\_VALUE (depending on your JDBC driver) as a String.  **Example**: persistence.xml file  <property name="eclipselink.jdbc.read-connections.max" value="3"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.JDBC\_READ\_CONNECTIONS\_MAX, "3"); | 2 |
| eclipselink.jdbc.read-connections.min | The minimum number of connections allowed in the JDBC read connection pool.1  Valid values: 0 to Integer.MAX\_VALUE (depending on your JDBC driver) as a String.  **Example**: persistence.xml file  <property name="eclipselink.jdbc.read-connections.min" value="1"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.JDBC\_READ\_CONNECTIONS\_MIN, "1"); | 2 |
| eclipselink.jdbc.read-connections.shared | Specify whether or not to allow concurrent use of shared read connections.1  The following are the valid values:   * true – allow concurrent use of shared read connections. * false – do not allow the concurrent use of shared read connections; concurrent readers are each allocated their own read connection.   **Example**: persistence.xml file  <property name="eclipselink.jdbc.read-connections.shared" value="true"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.JDBC\_READ\_CONNECTIONS\_SHARED, "true"); | false |
| eclipselink.jdbc.write-connections.max | The maximum number of connections allowed in the JDBC write connection pool.1  Valid values: to Integer.MAX\_VALUE (depending on your JDBC driver) as a String.  **Example**: persistence.xml file  <property name="eclipselink.jdbc.write-connections.max" value="5"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.JDBC\_WRITE\_CONNECTIONS\_MAX, "5"); | 10 |
| eclipselink.jdbc.write-connections.min | The maximum number of connections allowed in the JDBC write connection pool.1  Valid values: 0 to Integer.MAX\_VALUE (depending on your JDBC driver) as a String.  **Example**: persistence.xml file  <property name="eclipselink.jdbc.write-connections.min" value="2"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.JDBC\_WRITE\_CONNECTIONS\_MIN, "2"); | 5 |

1 This property applies when used in a Java SE environment.

For information about the use of annotations as opposed to persistence unit properties and vice versa, see the following:

* [What You May Need to Know About Overriding Annotations in JPA](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Overriding_Annotations_in_JPA)
* [What You May Need to Know About EclipseLink JPA Overriding Mechanisms](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_EclipseLink_JPA_Overriding_Mechanisms).

For more information about persistence unit properties, see [hat You May Need to Know About Using EclipseLink JPA Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Using_EclipseLink_JPA_Persistence_Unit_Properties).

**Using EclipseLink JPA Extensions for Logging**

This table lists the EclipseLink JPA persistence unit properties that you can define in a persistence.xml file to configure EclipseLink [logging](http://wiki.eclipse.org/Introduction_to_EclipseLink_Sessions_%28ELUG%29#Logging). Additional information (including examples) is available in [How to configure a custom logger in JPA](http://wiki.eclipse.org/EclipseLink/Examples/JPA/CustomLogger).

***EclipseLink JPA Persistence Unit Properties for Logging***

|  |  |  |
| --- | --- | --- |
| **Property** | **Usage** | **Default** |
| eclipselink.logging.logger | Select the type of logger to use.  The following are the valid values for the use in the persistence.xml file and for theorg.eclipse.persistence.config.PersistenceUnitProperties:   * DefaultLogger – the EclipseLink native logger org.eclipse.persistence.logging.DefaultSessionLog. * JavaLogger – the java.util.logging logger org.eclipse.persistence.logging.JavaLog. * ServerLogger – the java.util.logging logger org.eclipse.persistence.platform.server.ServerLog. Integrates with the application server's logging as define in the org.eclipse.persistence.platform.server.ServerPlatform. * Fully qualified class name of a custom logger. The custom logger must implement theorg.eclipse.persistence.logging.SessionLog interface.   **Example**: persistence.xml file  <property name="eclipselink.logging.logger" value="acme.loggers.MyCustomLogger" />  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.LOGGING\_LOGGER, "acme.loggers.MyCustomLogger" /> | DefaultLogger |
| eclipselink.logging.level | Control the amount and detail of log output by configuring the log level (in ascending order of information).  The following are the valid values for the use in the persistence.xml file and for the java.util.logging.Level:   * OFF – disables logging. You may want to set logging to OFF during production to avoid the overhead of logging. * SEVERE – logs exceptions indicating that EclipseLink cannot continue, as well as any exceptions generated during login. This includes a stack trace. * WARNING – logs exceptions that do not force EclipseLink 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. You may want to use the CONFIG log level at deployment time. * FINE – logs SQL. You may want to use this log level during debugging and testing, but not at production time. * FINER – similar to WARNING. Includes stack trace. You may want to use this log level during debugging and testing, but not at production time. * FINEST – includes additional low level information. You may want to use this log level during debugging and testing, but not at production time. * ALL – logs at the same level as FINEST.   **Example**: persistence.xml file  <property name="eclipselink.logging.level" value="OFF" />  **Example**: property Map  import java.util.logging.Level;  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.LOGGING\_LEVEL, Level.OFF); | Level.INFO |
| eclipselink.logging.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" />  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.LOGGING\_TIMESTAMP, "false"); | true |
| eclipselink.logging.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.   **Example**: persistence.xml file  <property name="eclipselink.logging.thread" value="false" />  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.LOGGING\_THREAD, "false"); | true |
| eclipselink.logging.session | Control whether an EclipseLink session identifier is logged in each log entry.  The following are the valid values:   * true – log an EclipseLink session identifier. * false – do not log an EclipseLink session identifier.   **Example**: persistence.xml file  <property name="eclipselink.logging.session" value="false" />  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.LOGGING\_SESSION, "false"); | true |
| eclipselink.logging.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" />  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.LOGGING\_EXCEPTIONS, "true"); | false |
| eclipselink.logging.file | Specify a file location for the log output (instead of the standard out).1  Valid values: a string location to a directory in which you have write access. The location may be relative to your current working directory or absolute.  **Example**: persistence.xml file  <property name="eclipselink.logging.file" value="C:\myout\" />  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.LOGGING\_FILE, "C:\myout\"); |  |

1 This property applies when used in a Java SE environment.

For information about the use of annotations as opposed to persistence unit properties and vice versa, see the following:

* [What You May Need to Know About Overriding Annotations in JPA](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Overriding_Annotations_in_JPA)
* [What You May Need to Know About EclipseLink JPA Overriding Mechanisms](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_EclipseLink_JPA_Overriding_Mechanisms).

For more information about persistence unit properties, see [What You May Need to Know About Using EclipseLink JPA Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Using_EclipseLink_JPA_Persistence_Unit_Properties).

**Using EclipseLink JPA Extensions for Session, Target Database and Target Application Server**

This table lists the EclipseLink JPA persistence unit properties that you can define in a persistence.xml file to configure EclipseLink extensions for session, as well as the target database and application server.

***EclipseLink JPA Persistence Unit Properties for Database, Session, and Application Server***

|  |  |  |
| --- | --- | --- |
| **Property** | **Usage** | **Default** |
| eclipselink.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 an 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"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.SESSION\_NAME, "MySession"); | EclipseLink-generated unique name. |
| eclipselink.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. For more information, see [hat You May Need to Know About EclipseLink JPA Overriding Mechanisms](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_EclipseLink_JPA_Overriding_Mechanisms).  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="eclipselink.sessions-xml" value="mysession.xml"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.SESSIONS\_XML, "mysession.xml"); |  |
| eclipselink.session-event-listener | Specify a descriptor event listener to be added during bootstrapping.  For more information, see [Obtaining an Entity Manager Factory in Java SE Environment](http://wiki.eclipse.org/Developing_Applications_Using_EclipseLink_JPA_%28ELUG%29#Obtaining_an_Entity_Manager_Factory_in_Java_SE_Environment).  Valid values: qualified class name for a class that implements theorg.eclipse.persistence.sessions.SessionEventListener interface.  **Example**: persistence.xml file  <property name="eclipselink.session-event-listener" value="mypackage.MyClass.class"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.SESSION\_EVENT\_LISTENER\_CLASS, "mypackage.MyClass.class"); |  |
| eclipselink.session.include.descriptor.queries | Enable or disable the default copying of all [named queries](http://wiki.eclipse.org/Introduction_to_EclipseLink_Queries_%28ELUG%29#Named_Queries) from the descriptors to the session. These queries include the ones defined using EclipseLink API, descriptor [amendment methods](http://wiki.eclipse.org/Introduction_to_Descriptors_%28ELUG%29#Amendment_and_After-Load_Methods), and so on.  The following are the valid values:   * true – enable the default copying of all named queries from the descriptors to the session. * false – disable the default copying of all named queries from the descriptors to the session.   **Example**: persistence.xml file  <property name="eclipselink.session.include.descriptor.queries" value="false"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.INCLUDE\_DESCRIPTOR\_QUERIES, "false"); | true |
| eclipselink.target-database | Specify the type of database that your JPA application uses.  The following are the valid values for the use in a persistence.xml file and for theorg.eclipse.persistence.config.TargetDatabase:   * Attunity – configure the persistence provider to use an Attunity database. * Auto – EclipseLink accesses the database and uses the metadata that JDBC provides to determine the target database. Applicable to JDBC drivers that support this metadata. * Cloudscape – configure the persistence provider to use a Cloudscape database. * Database – configure the persistence provider to use a generic choice if your target database is not listed here and your JDBC driver does not support the use of metadata that the Auto option requires. * DB2 – configure the persistence provider to use a DB2 database. * DB2Mainframe – configure the persistence provider to use a DB2Mainframe database. * DBase – configure the persistence provider to use a DBase database. * Derby – configure the persistence provider to use a Derby database. * HSQL – configure the persistence provider to use an HSQL database. * Informix – configure the persistence provider to use an Informix database. * JavaDB – configure the persistence provider to use a JavaDB database. * MySQL – configure the persistence provider to use a MySQL database. * Oracle – configure the persistence provider to use an Oracle Database. * PointBase – configure the persistence provider to use a PointBase database. * PostgreSQL – configure the persistence provider to use a PostgreSQL database. * SQLAnywhere – configure the persistence provider to use an SQLAnywhere database. * SQLServer – configure the persistence provider to use an SQLServer database. * Sybase – configure the persistence provider to use a Sybase database. * TimesTen – configure the persistence provider to use a TimesTen database.   You can also set the value to the fully qualified classname of a subclass of theorg.eclipse.persistence.platform.database.DatabasePlatform class.  **Example**: persistence.xml file  <property name="eclipselink.target-database" value="Oracle"/>  **Example**: persistence.xml file  <property name="eclipselink.target-database" value="org.eclipse.persistence.platform.database.HSQLPlatform"/>  **Example**: property Map  import org.eclipse.persistence.config.TargetDatabase;  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.TARGET\_DATABASE, TargetDatabase.Oracle); | Auto |
| eclipselink.target-server | Specify the type of application server that your JPA application uses.  The following are the valid values for the use in the persistence.xml file and theorg.eclipse.persistence.config.TargetServer:   * None – configure the persistence provider to use no application server. * WebLogic – configure the persistence provider to use Oracle WebLogic Server. Note: this server sets this property automatically, so you do not need to set it, unless it is disabled. * WebLogic\_9 – configure the persistence provider to use Oracle WebLogic Server version 9. * WebLogic\_10 – configure the persistence provider to use Oracle WebLogic Server version 10. * OC4J – configure the persistence provider to use OC4J. * SunAS9 – configure the persistence provider to use Sun Application Server version 9. Note: this server sets this property automatically, so you do not need to set it, unless it is disabled. * WebSphere – configure the persistence provider to use WebSphere Application Server. * WebSphere\_6\_1 – configure the persistence provider to use WebSphere Application Server version 6.1. * JBoss – configure the persistence provider to use JBoss Application Server. * NetWeaver\_7\_1 - configure the persistence provider to use SAP NetWeaver Application Server version 7.1 and higher. * Fully qualified class name of a custom server class that implementsorg.eclipse.persistence.platform.ServerPlatform interface.   **Example**: persistence.xml file  <property name="eclipselink.target-server" value="OC4J\_10\_1\_3"/>  **Example**: property Map  import org.eclipse.persistence.config.TargetServer;  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.TARGET\_SERVER, TargetServer.OC4J\_10\_1\_3); | None |

For information about the configuration of platforms, see the following:

* [Target Platforms](http://wiki.eclipse.org/Introduction_to_EclipseLink_Application_Development_%28ELUG%29#Target_Platforms)
* [Integrating EclipseLink with an Application Server](http://wiki.eclipse.org/Integrating_EclipseLink_with_an_Application_Server_%28ELUG%29)
* [Projects and Platforms](http://wiki.eclipse.org/Introduction_to_Projects_%28ELUG%29)
* [Database Platforms](http://wiki.eclipse.org/Introduction_to_Data_Access_%28ELUG%29)

For information about the use of annotations as opposed to persistence unit properties and vice versa, see the following:

* [What You May Need to Know About Overriding Annotations in JPA](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Overriding_Annotations_in_JPA)
* [What You May Need to Know About EclipseLink JPA Overriding Mechanisms](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_EclipseLink_JPA_Overriding_Mechanisms).

For more information about persistence unit properties, see [What You May Need to Know About Using EclipseLink JPA Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Using_EclipseLink_JPA_Persistence_Unit_Properties).

**Using EclipseLink JPA Extensions for Schema Generation**

This table lists the EclipseLink JPA persistence unit properties that you can define in a persistence.xml file to configure schema generation.

***EclipseLink JPA Persistence Unit Properties for Schema Generation***

|  |  |  |
| --- | --- | --- |
| **Property** | **Usage** | **Default** |
| eclipselink.ddl-generation | Specify what Data Definition Language (DDL) generation action you want for your JPA entities. To specify the DDL generation target, see [eclipselink.ddl-generation.output-mode](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.ddl-generation.output-mode).  The following are the valid values for the use in a persistence.xml file:   * none – EclipseLink does not generate DDL; no schema is generated. * create-tables – EclipseLink will attempt to execute a CREATE TABLE SQL for each table. If the table already exists, EclipseLink will follow the default behavior of your specific database and JDBC driver combination (when a CREATE TABLE SQL is issued for an already existing table). In most cases an exception is thrown and the table *is not* created. EclipseLink will then continue with the next statement. (See also [eclipselink.create-ddl-jdbc-file-name](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.create-ddl-jdbc-file-name).) * drop-and-create-tables – EclipseLink will attempt to DROP all tables, then CREATE all tables. If any issues are encountered, EclipseLink will follow the default behavior of your specific database and JDBC driver combination, then continue with the next statement. (See also [eclipselink.create-ddl-jdbc-file-name](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.create-ddl-jdbc-file-name) and [eclipselink.drop-ddl-jdbc-file-name](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.drop-ddl-jdbc-file-name).)   The following are the valid values for the org.eclipse.persistence.config.PersistenceUnitProperties:   * NONE – see none. * CREATE\_ONLY – see create-tables. * DROP\_AND\_CREATE – see drop-and-create-tables.   If you are using persistence in a Java SE environment and would like to create the DDL files without creating tables, additionally define a Java system property INTERACT\_WITH\_DB and set its value to false.  **Example**: persistence.xml file  <property name="eclipselink.ddl-generation" value="create-tables"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.DDL\_GENERATION, PersistenceUnitProperties.CREATE\_ONLY); | none or PersistenceUnitProperties.NONE |
| eclipselink.application-location | Specify where EclipseLink should write generated DDL files (see [eclipselink.create-ddl-jdbc-file-name](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.create-ddl-jdbc-file-name) and [eclipselink.drop-ddl-jdbc-file-name](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.drop-ddl-jdbc-file-name)). Files are written if [eclipselink.ddl-generation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.ddl-generation) is set to anything other than none.  Valid values: 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, EclipseLink will append one valid for your operating system.  **Example**: persistence.xml file  <property name="eclipselink.application-location" value="C:\ddl\"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.APP\_LOCATION, "C:\ddl\"); | "."+File.separator or <tt>PersistenceUnitProperties.DEFAULT\_APP\_LOCATION</tt> |
| eclipselink.create-ddl-jdbc-file-name | Specify the file name of the DDL file that EclipseLink generates containing SQL statements to create tables for JPA entities. This file is written to the location specified by [eclipselink.application-location](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.application-location) when [eclipselink.ddl-generation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.ddl-generation) is set to create-tablesor drop-and-create-tables.  Valid values: a file name valid for your operating system. Optionally, you may prefix the file name with a file path as long as the concatenation of [eclipselink.application-location](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.application-location) + [eclipselink.create-ddl-jdbc-file-name](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.create-ddl-jdbc-file-name) is a valid file specification for your operating system.  **Example**: persistence.xml file  <property name="eclipselink.create-ddl-jdbc-file-name" value="create.sql"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.CREATE\_JDBC\_DDL\_FILE, "create.sql"); | createDDL.jdbc orPersistenceUnitProperties.DEFAULT\_CREATE\_JDBC\_FILE\_NAME |
| eclipselink.drop-ddl-jdbc-file-name | Specify the file name of the DDL file that EclipseLink generates containing the SQL statements to drop tables for JPA entities. This file is written to the location specified by [eclipselink.application-location](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.application-location) when [eclipselink.ddl-generation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.ddl-generation) is set to drop-and-create-tables.  Valid values: a file name valid for your operating system. Optionally, you may prefix the file name with a file path as long as the concatenation of [eclipselink.application-location](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.application-location) + [eclipselink.drop-ddl-jdbc-file-name](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.drop-ddl-jdbc-file-name) is a valid file specification for your operating system.  **Example**: persistence.xml file  <property name="eclipselink.drop-ddl-jdbc-file-name" value="drop.sql"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.DROP\_JDBC\_DDL\_FILE, "drop.sql"); | dropDDL.jdbc orPersistenceUnitProperties.DEFAULT\_DROP\_JDBC\_FILE\_NAME |
| eclipselink.ddl-generation.output-mode | Use this property to specify the DDL generation target.  The following are the valid values for the use in the persistence.xml file:   * both – generate SQL files and execute them on the database. If [eclipselink.ddl-generation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.ddl-generation) is set to create-tables, then [eclipselink.create-ddl-jdbc-file-name](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.create-ddl-jdbc-file-name) is written to[eclipselink.application-location](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.application-location) and executed on the database. If [eclipselink.ddl-generation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.ddl-generation) is set to drop-and-create-tables, then both [eclipselink.create-ddl-jdbc-file-name](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.create-ddl-jdbc-file-name) and[eclipselink.drop-ddl-jdbc-file-name](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.drop-ddl-jdbc-file-name) are written to [eclipselink.application-location](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.application-location) and both SQL files are executed 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). If [eclipselink.ddl-generation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.ddl-generation) is set to create-tables, then [eclipselink.create-ddl-jdbc-file-name](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.create-ddl-jdbc-file-name) is written to[eclipselink.application-location](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.application-location). It is not executed on the database. If [eclipselink.ddl-generation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.ddl-generation) is set to drop-and-create-tables, then both [eclipselink.create-ddl-jdbc-file-name](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.create-ddl-jdbc-file-name) and[eclipselink.drop-ddl-jdbc-file-name](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.drop-ddl-jdbc-file-name) are written to [eclipselink.application-location](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.application-locatio). Neither is executed on the database.   The following are the valid values for the org.eclipse.persistence.config.PersistenceUnitProperties:   * DDL\_BOTH\_GENERATION – see both. * DDL\_DATABASE\_GENERATION – see database. * DDL\_SQL\_SCRIPT\_GENERATION – see sql-script.   **Example**: persistence.xml file  <property name="eclipselink.ddl-generation.output-mode" value="database"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.DDL\_GENERATION\_MODE, PersistenceUnitProperties.DDL\_DATABASE\_GENERATION); | Container or Java EE mode: database  Note: This setting may be overridden by containers with specific EclipseLink support. Please, refer to your container documentation for details.  Bootstrap or Java SE mode: both orPersistenceUnitProperties.DDL\_BOTH\_GENERATION |

For information about the use of annotations as opposed to persistence unit properties and vice versa, see the following:

* [What You May Need to Know About Overriding Annotations in JPA](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Overriding_Annotations_in_JPA)
* [What You May Need to Know About EclipseLink JPA Overriding Mechanisms](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_EclipseLink_JPA_Overriding_Mechanisms).

For more information about persistence unit properties, see [What You May Need to Know About Using EclipseLink JPA Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Using_EclipseLink_JPA_Persistence_Unit_Properties).

**Using EclipseLink JPA Extensions for Tracking Changes**

Within a transaction, EclipseLink automatically tracks entity changes.

EclipseLink defines one annotation for tracking changes – [@ChangeTracking](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40ChangeTracking_Annotation) annotation.

EclipseLink also provides a number of persistence unit properties that you can specify to configure EclipseLink change tracking (see [How to Use the Persistence Unit Properties for Change Tracking](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_Persistence_Unit_Properties_for_Change_Tracking)). These properties may compliment or provide an alternative to the usage of annotations.

|  |
| --- |
| **Note:** Persistence unit properties always override the corresponding annotations' attributes. For more information, see [What You May Need to Know About Overriding Annotations in JPA](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Overriding_Annotations_in_JPA) and [What You May Need to Know About Using EclipseLink JPA Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Using_EclipseLink_JPA_Persistence_Unit_Properties). |

For more information, see [Unit of Work and Change Policy](http://wiki.eclipse.org/Introduction_to_EclipseLink_Transactions_%28ELUG%29#Unit_of_Work_and_Change_Policy).

**How to Use the @ChangeTracking Annotation**

Use the [@ChangeTracking](http://www.eclipse.org/eclipselink/api/1.0/org/eclipse/persistence/annotations/ChangeTracking.html) annotation to set the unit of work's change policy.

@Target({TYPE})

@Retention(RUNTIME)

public @interface ChangeTracking {

ChangeTrackingType value() default AUTO;

}

|  |
| --- |
| **Note:** This is an optimization feature that lets you tune the way EclipseLink detects changes. You should choose the strategy based on the usage and data modification patterns of the entity type as different types may have different access patterns and hence different settings, and so on.  For more information, see the following:   * [Optimizing the Unit of Work](http://wiki.eclipse.org/Optimizing_the_EclipseLink_Application_%28ELUG%29#Optimizing_the_Unit_of_Work) * [Read Optimization Examples](http://wiki.eclipse.org/Optimizing_the_EclipseLink_Application_%28ELUG%29#Read_Optimization_Examples) * [Write Optimization Examples](http://wiki.eclipse.org/Optimizing_the_EclipseLink_Application_%28ELUG%29#Write_Optimization_Examples) |

This table lists attributes of the @ChangeTracking annotation.

***Attributes of the @ChangeTracking Annotation***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| value | Set this attribute to the type of the change tracking (org.eclipse.persistence.annotations.ChangeTrackingTypeenumerated type) to use.  The following are the valid values for the ChangeTrackingType:   * ATTRIBUTE – This option uses weaving to detect which fields or properties of the object change. This is the most efficient option, but you must enable weaving to use it.  For more information, see [Attribute Change Tracking Policy](http://wiki.eclipse.org/Introduction_to_EclipseLink_Transactions_%28ELUG%29#Attribute_Change_Tracking_Policy). * OBJECT – This option uses weaving to detect if the object has been changed, but uses a backup copy of the object to determine what fields or properties changed. This is more efficient than the DEFERRED option, but less efficient than theATTRIBUTE option. You must enable weaving to use this option. This option supports additional mapping configuration to attribute.  For more information, see [Object-Level Change Tracking Policy](http://wiki.eclipse.org/Introduction_to_EclipseLink_Transactions_%28ELUG%29#Object-Level_Change_Tracking_Policy). * DEFERRED – This option defers all change detection to the UnitOfWork's change detection process. This option uses a backup copy of the object to determine which fields or properties changed. This is the least efficient option, but does not require weaving and supports additional mapping configurations to attribute and object.  For more information, see [Deferred Change Detection Policy](http://wiki.eclipse.org/Introduction_to_EclipseLink_Transactions_%28ELUG%29#Deferred_Change_Detection_Policy). * AUTO – This option does not set any change tracking policy. The policy is determined by the EclipseLink agent: if the class can be weaved for change tracking the ATTRIBUTE option is used; otherwise, the DEFERRED option is used. AUTO is the only change tracking value where EclipseLink chooses the value; any other value will explicitly cause EclipseLink to use that value, so if you decide to explicitly set it, you must set it correctly and use your model correctly to ensure the change tracking detects your changes.   Note: For every option, objects with changed attributes will be processed at the commit time to include any changes in the results of the commit. Unchanged objects will be ignored.  Note: The weaving of change tracking is the same for attribute and object change tracking.  For information about the relationship between the value attribute and eclipselink.weaving.changetracking property, see[What You May Need to Know About the Relationship Between the Change Tracking Annotation and Persistence Unit Property](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_the_Relationship_Between_the_Change_Tracking_Annotation_and_Persistence_Unit_Property) | ChangeTrackingType.AUTO | optional |

This example shows how to use the @ChangeTracking annotation.

***Usage of @ChangeTracking Annotation***

@Entity

@Table(name="EMPLOYEE")

@ChangeTracking(OBJECT) (

public class Employee implements Serializable {

...

}

|  |
| --- |
| **Note:** You cannot use the attribute change tracking in conjunction with the following:   * optimistic field-level locking (see [Optimistic Locking](http://wiki.eclipse.org/Introduction_to_EclipseLink_Application_Development_%28ELUG%29#Optimistic_Locking)); * mutable types (such as mutable temporals or mutable serialized mappings); * non-lazy collection mappings.   The attribute change tracking will not detect object changes made through reflection or direct field access to fields mapped as properties. |

**How to Use the Persistence Unit Properties for Change Tracking**

This table lists the persistence unit properties that you can define in a persistence.xml file to configure EclipseLink change tracking.

***EclipseLink JPA Persistence Unit Properties for Change Tracking***

|  |  |  |
| --- | --- | --- |
| **Property** | **Usage** | **Default** |
| eclipselink.weaving.changetracking | Enable or disable the AttributeLevelChangeTracking through weaving.  The following are the valid values:   * true – enable the AttributeLevelChangeTracking through weaving. When enabled, only classes with all mappings allowing change tracking have change tracking enabled. * false – disable the AttributeLevelChangeTracking through weaving. Use this setting if the following applies:   + you cannot weave at all;   + you do not want your classes to be changed during weaving (for example, for debugging purposes);   + you wish to disable this feature for configurations that do not support it (for example, you are mutating thejava.util.Date or java.util.Calendar, you are using property access but modifying the underlying instance variables).   Note: you may set this option only if the [eclipselink.weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.weaving) option is set to true. The purpose of theeclipselink.weaving.changetracking option is to provide more granular control over weaving.  For more information, see the following:   * [Using EclipseLink JPA Weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_EclipseLink_JPA_Weaving) * [Configuring Change Policy](http://wiki.eclipse.org/Configuring_a_Descriptor_%28ELUG%29#Configuring_Change_Policy)   **Example**: persistence.xml file  <property name="eclipselink.weaving.changetracking" value="false"/>  **Example**: property Map  import org.eclipse.persistence.config.PersistenceUnitProperties;  propertiesMap.put(PersistenceUnitProperties.WEAVING\_CHANGE\_TRACKING, "false"); | true |

For information about the relationship between the value attribute of the [@ChangeTracking](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40ChangeTracking_Annotation) annotation and eclipselink.weaving.changetracking property, see [What You May Need to Know About the Relationship Between the Change Tracking Annotation and Persistence Unit Property](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_the_Relationship_Between_the_Change_Tracking_Annotation_and_Persistence_Unit_Property).

For information about the use of annotations as opposed to persistence unit properties and vice versa, see the following:

* [What You May Need to Know About Overriding Annotations in JPA](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Overriding_Annotations_in_JPA)
* [What You May Need to Know About EclipseLink JPA Overriding Mechanisms](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_EclipseLink_JPA_Overriding_Mechanisms).

For more information about persistence unit properties, see [What You May Need to Know About Using EclipseLink JPA Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Using_EclipseLink_JPA_Persistence_Unit_Properties).

**What You May Need to Know About the Relationship Between the Change Tracking Annotation and Persistence Unit Property**

The following table shows the dependency between the value attribute of the [@ChangeTracking](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_.40ChangeTracking_Annotation) annotation and the eclipselink.weaving.changetracking property (see [How to Use the Persistence Unit Properties for Change Tracking](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_the_Persistence_Unit_Properties_for_Change_Tracking)).

***Relationship Between the Change Tracking Annotation and Property***

|  |  |  |
| --- | --- | --- |
| **@ChangeTracking(value=)** | **eclipselink.weaving.changetracking=** | **Description** |
| ATTRIBUTE | true | Weave and enable attribute change tracking.  Even if the descriptors contain mappings that are not directly supported by change tracking weaving, EclipseLink assumes that you will either use your set methods to ensure that your changes are captured, or raise the events for these mappings. |
| ATTRIBUTE | false | Do not weave change tracking.  You must implement the org.eclipse.persistence.annotations.ChangeTracking interface and raise the change events. Otherwise, an error will occur on descriptor initialization. |
| OBJECT | true | Object change tracking is used and change tracking is woven.  Note: Weaving is identical for object and attribute change tracking. |
| OBJECT | false | Do not weave change tracking.  You must implement the org.eclipse.persistence.annotations.ChangeTracking interface and raise the change events. Otherwise, an error will occur on descriptor initialization. |
| DEFERRED | true | Do not weave change tracking.  Deferred change tracking is used. |
| DEFERRED | false | Do not weave change tracking.  Deferred change tracking is used. |
| AUTO | true | Weave change tracking if the descriptor does not use any mappings that do not support change tracking. |
| AUTO | false | Do not weave change tracking.  Deferred change tracking is used. |

**Using EclipseLink JPA Query Customization Extensions**

This section describes the following:

* [How to Use EclipseLink JPA Query Hints](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_EclipseLink_JPA_Query_Hints)
* [How to Use EclipseLink Query API in JPA Queries](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Use_EclipseLink_Query_API_in_JPA_Queries)

**How to Use EclipseLink JPA Query Hints**

The [EclipseLink JPA Query Hints](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Table_19-30) table lists the EclipseLink JPA query hints that you can specify when you construct a JPA query, as the [Specifying an EclipseLink JPA Query Hint](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-24) example shows, or when you specify a JPA query using the @QueryHint annotation (see Section 8.3.1 "NamedQuery Annotation" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220)), as the[Specifying an EclipseLink JPA Query Hint with @QueryHint](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-25) example shows.

All EclipseLink query hints are defined in the QueryHints class in the org.eclipse.persistence.config package.

The EclipseLink query hints include the following:

* [eclipselink.cache-usage](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Cache_Usage)
* [eclipselink.query-type](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Query_Type)
* [eclipselink.jdbc.bind-parameters](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Bind_Parameters)
* [eclipselink.pessimistic-lock](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Pessimistic_Lock)
* [eclipselink.refresh](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Refresh)
* [eclipselink.refresh.cascade](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Refresh)
* [eclipselink.maintain-cache](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Maintain_Cache)
* [eclipselink.batch](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Batch)
* [eclipselink.join-fetch](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Join_Fetch)
* [eclipselink.read-only](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Read_Only)
* [eclipselink.jdbc.timeout](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Timeout)
* [eclipselink.jdbc.fetch-size](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Fetch_Size)
* [eclipselink.jdbc.max-rows](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Max_Rows)
* [eclipselink.result-collection-type](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Result_Collection_Type)

When you set a hint, you can set the value using the public static final field in the appropriate configuration class in org.eclipse.persistence.config package, including the following:

* HintValues
* CacheUsage
* PessimisticLock
* QueryType

The [Specifying an EclipseLink JPA Query Hint](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-24) and [Specifying an EclipseLink JPA Query Hint with @QueryHint](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-25) examples show how to set the value of hinteclipselink.jdbc.bind-parameters using the QueryHints configuration class to set the name of the hint, and the HintValues configuration class to set the value.

***Specifying an EclipseLink JPA Query Hint***

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

Customer customer = (Customer)entityMgr.createNamedQuery("findCustomerBySSN").

setParameter("SSN", "123-12-1234").

setHint(QueryHints.BIND\_PARAMETERS, HintValues.PERSISTENCE\_UNIT\_DEFAULT).

getSingleResult();

***Specifying an EclipseLink JPA Query Hint with @QueryHint***

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

@Entity

@NamedQuery(

name="findEmployeeByDept",

query="SELECT e FROM Employee e WHERE e.dept=:deptNum",

hints=@QueryHint(name=QueryHints.BIND\_PARAMETERS, value=HintValues.TRUE)

)

public class Employee implements Serializable {

...

}

**Cache Usage**

The eclipselink.cache-usage hint specifies how the query should interact with the EclipseLink cache.

EclipseLink JPA uses a shared cache mechanism that is scoped to the entire persistence unit. When operations are completed in a particular persistence context, the results are merged back into the shared cache so that other persistence contexts can use them. This happens regardless of whether the entity manager and persistence context are created in Java SE or Java EE. Any entity persisted or removed using the entity manager will always be kept consistent with the cache.

For more information, see the following:

* [Session Cache](http://wiki.eclipse.org/Introduction_to_Cache_%28ELUG%29#Session_Cache)
* [How to Use In-Memory Queries](http://wiki.eclipse.org/Introduction_to_EclipseLink_Queries_%28ELUG%29#How_to_Use_In-Memory_Queries)

The following are the valid values for the org.eclipse.persistence.config.CacheUsage:

* DoNotCheckCache – Always go to the database.
* CheckCacheByExactPrimaryKey – If a read-object query contains an expression where the primary key is the only comparison, you can obtain a cache hit if you process the expression against the object in memory
* CheckCacheByPrimaryKey – If a read-object query contains an expression that compares at least the primary key, you can obtain a cache hit if you process the expression against the objects in memory.
* CheckCacheThenDatabase – You can configure any read-object query to check the cache completely before you resort to accessing the database.
* CheckCacheOnly – You can configure any read-all query to check only the parent session cache (shared cache) and return the result from it without accessing the database.
* ConformResultsInUnitOfWork – You can configure any read-object or read-all query within the context of a unit of work to conform the results with the changes to the object made within that unit of work. This includes new objects, deleted objects and changed objects.  
  For more information, see [Using Conforming Queries and Descriptors](http://wiki.eclipse.org/Using_Advanced_Unit_of_Work_API_%28ELUG%29#Using_Conforming_Queries_and_Descriptors).
* UseEntityDefault – Use the cache configuration as specified by the EclipseLink descriptor API for this entity.  
  Note: the entity default value is to not check the cache (DoNotCheckCache). The query will access the database and synchronize with the cache. Unless refresh has been set on the query, the cached objects will be returned without being refreshed from the database. EclipseLink does not support the cache usage for native queries or queries that have complex result sets such as returning data or multiple objects.

Default: CacheUsage.UseEntityDefault  
Note: this default is DoNotCheckCache.

For more information, see [Configuring Cache Usage for In-Memory Queries](http://wiki.eclipse.org/Introduction_to_EclipseLink_Queries_%28ELUG%29#Configuring_Cache_Usage_for_In-Memory_Queries).

**Example**: JPA Query API

import org.eclipse.persistence.config.CacheUsage;

import org.eclipse.persistence.config.QueryHints;

query.setHint(QueryHints.CACHE\_USAGE, CacheUsage.CheckCacheOnly);

**Example**: @QueryHint

import org.eclipse.persistence.config.CacheUsage;

import org.eclipse.persistence.config.TargetDatabase;

@QueryHint(name=QueryHints.CACHE\_USAGE, value=CacheUsage.CheckCacheOnly);

**Query Type**

The eclipselink.query-type hint specifies the EclipseLink query type to use for the query.

For most JP QL queries, the org.eclipse.persistence.queries.ReportQuery or org.eclipse.persistence.queries.ReadAllQuery are used. The eclipselink.query-typehint lets you use other query types, such as org.eclipse.persistence.queries.ReadObjectQuery for queries that are know to return a single object.

For more information, see the following:

* [Object-Level Read Query](http://wiki.eclipse.org/Introduction_to_EclipseLink_Queries_%28ELUG%29#Object-Level_Read_Query)
* [Data-Level Modify Query](http://wiki.eclipse.org/Introduction_to_EclipseLink_Queries_%28ELUG%29#Data-Level_Modify_Query)

The following are the valid values for the org.eclipse.persistence.config.QueryType:

* Auto – EclipseLink chooses the type of query to use.
* ReadAll – Use the [ReadAllQuery](http://wiki.eclipse.org/Introduction_to_EclipseLink_Queries_%28ELUG%29#ReadAllQuery) type for the query.
* ReadObject – Use the [ReadObjectQuery](http://wiki.eclipse.org/Introduction_to_EclipseLink_Queries_%28ELUG%29) type for the query.
* Report – Use the [Report Query](http://wiki.eclipse.org/Introduction_to_EclipseLink_Queries_%28ELUG%29#Report_Query) type for the query.

Default: QueryType.Auto

**Example**: JPA Query API

import org.eclipse.persistence.config.QueryType;

import org.eclipse.persistence.config.QueryHints;

query.setHint(QueryHints.QUERY\_TYPE, QueryType.ReadObject);

**Example**: @QueryHint

import org.eclipse.persistence.config.QueryType;

import org.eclipse.persistence.config.TargetDatabase;

@QueryHint(name=QueryHints.QUERY\_TYPE, value=QueryType.ReadObject);

**Bind Parameters**

The eclipselink.jdbc.bind-parameters hint controls whether or not the query uses parameter binding.

For more information, see [How to Use Parameterized SQL (Parameter Binding) and Prepared Statement Caching for Optimization](http://wiki.eclipse.org/Optimizing_the_EclipseLink_Application_%28ELUG%29#How_to_Use_Parameterized_SQL_.28Parameter_Binding.29_and_Prepared_Statement_Caching_for_Optimization).

The following are the valid values for the org.eclipse.persistence.config.HintValues:

* TRUE – bind all parameters.
* FALSE – do not bind all parameters.
* PERSISTENCE\_UNIT\_DEFAULT – use the parameter binding setting made in your EclipseLink session's database login, which is true by default.  
    
  For more information, see [Configuring JDBC Options](http://wiki.eclipse.org/Configuring_a_Database_Login_%28ELUG%29#Configuring_JDBC_Options).

Default: HintValues.PERSISTENCE\_UNIT\_DEFAULT

**Example**: JPA Query API

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

query.setHint(QueryHints.BIND\_PARAMETERS, HintValues.TRUE);

**Example**: @QueryHint

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.TargetDatabase;

@QueryHint(name=QueryHints.BIND\_PARAMETERS, value=HintValues.TRUE);

**Fetch Size**

The eclipselink.jdbc.fetch-size hint specifies the number of rows that should be fetched from the database when more rows are needed1

For large queries that return a large number of objects you can configure the row fetch size used in the query to improve performance by reducing the number database hits required to satisfy the selection criteria. Most JDBC drivers default to a fetch size of 10, so if you are reading 1000 objects, increasing the fetch size to 256 can significantly reduce the time required to fetch the query's results. The optimal fetch size is not always obvious. Usually, a fetch size of one half or one quarter of the total expected result size is optimal. Note that if you are unsure of the result set size, incorrectly setting a fetch size too large or too small can decrease performance.

A value of 0 means the JDBC driver default will be used.

Valid values: 0 to Integer.MAX\_VALUE (depending on your JDBC driver) as a String.

Default: 0

Note: this value indicates that the JDBC driver default will be used.

1 This property is dependent on the JDBC driver support.

**Timeout**

The eclipselink.jdbc.timeout hint specifies the number of seconds EclipseLink will wait on a query before throwing a DatabaseException1

A value of 0 means EclipseLink will never time-out a query.

Valid values: 0 to Integer.MAX\_VALUE (depending on your JDBC driver) as a String.

Default: 0

1 This property is dependent on the JDBC driver support.

**Pessimistic Lock**

The eclipselink.pessimistic-lock hint controls whether or not pessimistic locking is used.

The following are the valid values for the org.eclipse.persistence.config.PessimisticLock:

* NoLock – pessimistic locking is not used.
* Lock – EclipseLink issues a SELECT .... FOR UPDATE.
* LockNoWait – EclipseLink issues a SELECT .... FOR UPDATE NO WAIT.

Default: NoLock

**Example**: JPA Query API

import org.eclipse.persistence.config.PessimisticLock;

import org.eclipse.persistence.config.QueryHints;

query.setHint(QueryHints.PESSIMISTIC\_LOCK, PessimisticLock.LockNoWait);

**Example**: @QueryHint

import org.eclipse.persistence.config.PessimisticLock;

import org.eclipse.persistence.config.QueryHints;

@QueryHint(name=QueryHints.PESSIMISTIC\_LOCK, value=PessimisticLock.LockNoWait);

**Batch**

The eclipselink.batch hint supplies EclipseLink with batching information so subsequent queries of related objects can be optimized in batches instead of being retrieved one-by-one or in one large joined read. Batch reading is more efficient than joining because it avoids reading duplicate data.

Batching is only allowed on queries that have a single object in their select clause.

Valid values: a single-valued relationship path expression.

Note: use dot notation to access nested attributes. For example, to batch-read an employee's manager's address, specify e.manager.address

**Example**: JPA Query API

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

query.setHint("eclipselink.batch", "e.address");

**Example**: @QueryHint

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

@QueryHint(name=QueryHints.BATCH, value="e.address");

**Join Fetch**

The eclipselink.join-fetch hint allows joining of the attributes.

This is similar to eclipselink.batch: subsequent queries of related objects can be optimized in batches instead of being retrieved in one large joined read.

This is different from JP QL joining because it allows multilevel fetch joins.

For more information, see Section 4.4.5.3 "Fetch Joins" of JPA specification.

Valid values: a relationship path expression.

Note: use dot notation to access nested attributes.

**Example**: JPA Query API

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

query.setHint("eclipselink.join-fetch", "e.address");

**Example**: @QueryHint

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

@QueryHint(name=QueryHints.FETCH, value="e.address");

**Refresh**

The eclipselink.refresh hint controls whether or not to update the EclipseLink session cache with objects that the query returns.

The following are the valid values for the org.eclipse.persistence.config.HintValues:

* TRUE – refresh cache.
* FALSE – do not refresh cache.

Default: FALSE

**Example**: JPA Query API

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

query.setHint(QueryHints.REFRESH, HintValues.TRUE);

**Example**: @QueryHint

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

@QueryHint(name=QueryHints.REFRESH, value=HintValues.TRUE);

**Maintain Cache**

The eclipselink.maintain-cache hint controls whether or not query results are cached in the session cache and provides a way to query the current database contents without affecting the current persistence context. It configures the query to return unmanaged instances so any updates to entities queried using this hint would have to be merged into the persistence context.

The following are the valid values for the org.eclipse.persistence.config.HintValues:

* TRUE – maintain cache.
* FALSE – do not maintain cache.

Default: FALSE

**Example**: JPA Query API

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

query.setHint(QueryHints.MAINTAIN\_CACHE, HintValues.FALSE);

**Example**: @QueryHint

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

@QueryHint(name=QueryHints.MAINTAIN\_CACHE, value=HintValues.FALSE);

**Read Only**

The eclipselink.read-only hint retrieves read-only results back from the query: on nontransactional read operations, where the requested entity types are stored in the shared cache, you can request that the shared instance be returned instead of a detached copy.

Note: you should never modify objects returned from the shared cache.

The following are the valid values for the org.eclipse.persistence.config.HintValues:

* TRUE – retrieve read-only results back from the query;
* FALSE – do not retrieve read-only results back from the query.

Default: FALSE

**Example**: JPA Query API

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

query.setHint(QueryHints.READ\_ONLY, HintValues.TRUE);

**Example**: @QueryHint

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

@QueryHint(name=QueryHints.READ\_ONLY, value=HintValues.TRUE);

**Result Collection Type**

The eclipselink.result-collection-type hint configures the concrete class that EclipseLink should use to return its query result.

This lets you specify the type of collection in which the result will be returned.

Valid values: Java Class that implements the Collection interface.

Note: typically, you would execute these queries by calling the getResultsList method, which returns the java.util.List, on the Query. This means that the class specified in this hint must implement the List interface, if you are invoking it using the getResultsList method.

Note: specify the class without the ".class" notation. For example, java.util.Vector would work, not java.util.Vector.classEclipseLink will throw an exception, if you use this hint with a class that does not implement the Collection interface.

Default:java.util.Vector

**Example**: JPA Query API

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

query.setHint("eclipselink.result-collection-type", java.util.ArrayList.class);

**Example**: @QueryHint

import org.eclipse.persistence.config.HintValues;

import org.eclipse.persistence.config.QueryHints;

@QueryHint(name=QueryHints.RESULT\_COLLECTION\_TYPE, value="java.util.ArrayList");

**How to Use EclipseLink Query API in JPA Queries**

EclipseLink JPA provides an EclipseLink implementation class for each JPA persistence interface. By casting to the EclipseLink implementation class, you have full access to EclipseLink functionality.

This section provides the following examples:

* [Creating a JPA Query Using the EclipseLink Expressions Framework](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Creating_a_JPA_Query_Using_the_EclipseLink_Expressions_Framework)
* [Creating a JPA Query Using an EclipseLink DatabaseQuery](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Creating_a_JPA_Query_Using_an_EclipseLink_DatabaseQuery)
* [Creating a JPA Query Using an EclipseLink Call Object](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Creating_a_JPA_Query_Using_an_EclipseLink_Call_Object)
* [Using Named Parameters in a Native Query](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_Named_Parameters_in_a_Native_Query)
* [Using Java Persistence Query Language Positional Parameters in a Native Query](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_Java_Persistence_Query_Language_Positional_Parameters_in_a_Native_Query)
* [Using JDBC-Style Positional Parameters in a Native Query](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Using_JDBC-Style_Positional_Parameters_in_a_Native_Query)

For more information, see [Queries](http://wiki.eclipse.org/EclipseLink/UserGuide/Queries_%28ELUG%29).

**Creating a JPA Query Using the EclipseLink Expressions Framework**

EclipseLink provides an expression framework with which you can express queries in a database-neutral fashion as an alternative to raw SQL.

This example shows how to cast an entity manager to access an EclipseLink persistence provider createQuery method that takes an EclipseLink Expression.

***Creating a Query with the EclipseLink Expressions Framework***

((org.eclipse.persistence.jpa.JpaEntityManager)entityManager.getDelegate()).

createQuery(Expression expression, Class resultType);

EclipseLink expressions offer the following advantages over SQL when you access a database:

* Expressions are easier to maintain because the database is abstracted.
* Changes to descriptors or database tables do not affect the querying structures in the application.
* Expressions enhance readability by standardizing the Query interface so that it looks similar to traditional Java calling conventions.  
  For example, the Java code required to get the street name from the Address object of the Employee class looks like this:

emp.getAddress().getStreet().equals("Meadowlands");

The expression to get the same information is similar:

emp.get("address").get("street").equal("Meadowlands");

* Expressions allow read queries to transparently query between two classes that share a relationship. If these classes are stored in multiple tables in the database, EclipseLink automatically generates the appropriate join statements to return information from both tables.
* Expressions simplify complex operations.  
  For example, the following Java code retrieves all employees that live on "Meadowlands" whose salary is greater than 10,000:

ExpressionBuilder emp = new ExpressionBuilder();

Expression exp = emp.get("address").get("street").equal("Meadowlands");

Vector employees = session.readAllObjects(Employee.class,

exp.and(emp.get("salary").greaterThan(10000)));

EclipseLink automatically generates the appropriate SQL from the preceding code:

SELECT t0.VERSION, t0.ADDR\_ID, t0.EMP\_ID, t0.SALARY FROM EMPLOYEE t0, ADDRESS t1 WHERE (((t1.STREET = 'Meadowlands')AND (t0.SALARY > 10000)) AND (t1.ADDRESS\_ID = t0.ADDR\_ID))

For more information, see [Introduction to EclipseLink Expressions](http://wiki.eclipse.org/Introduction_to_EclipseLink_Expressions_%28ELUG%29).

**Creating a JPA Query Using an EclipseLink DatabaseQuery**

An EclipseLink [DatabaseQuery](http://wiki.eclipse.org/Introduction_to_EclipseLink_Queries_%28ELUG%29#DatabaseQuery) is a query object that provides a rich API for handling a variety of database query requirements, including reading and writing at the object level and at the data level.

This example shows how to cast a JPA query from an entity manager to access an EclipseLink persistence provider setDatabaseQuery method that takes an EclipseLinkDatabaseQuery.

***DatabaseQuery***

((org.eclipse.persistence.jpa.JpaQuery)query).setDatabaseQuery(DatabaseQuery query);

The following example shows how to cast a JPA query from an entity manager to access an EclipseLink persistence provider setDatabaseQuery method that takes an EclipseLink DataReadQuery initialized with an EclipseLink SQLCall object that specifies a SELECT. This query will return one or more objects.

***DatabaseQuery with Selecting Call***

((org.eclipse.persistence.jpa.JpaQuery)query).

setDatabaseQuery(new DataReadQuery(new SQLCall("SELECT...")));

The following example shows how to cast a JPA query from an entity manager to access an EclipseLink persistence provider setDatabaseQuery method that takes an EclipseLink DataModifyQuery initialized with an EclipseLink SQLCall object that specifies an UPDATE. This query will modify one or more objects; however, this query will not update the managed objects within the persistence context.

***DatabaseQuery with Non-Selecting Call***

((org.eclipse.persistence.jpa.JpaQuery)query).

setDatabaseQuery(new DataModifyQuery(new SQLCall("UPDATE...")));

**Creating a JPA Query Using an EclipseLink Call Object**

Using [DatabaseQuery](http://wiki.eclipse.org/Introduction_to_EclipseLink_Queries_%28ELUG%29#DatabaseQuery) method setCall, you can define your own EclipseLink Call to accommodate a variety of data source options, such as SQL stored procedures and stored functions, EJB QL queries, and EIS interactions.

This example shows how to cast a JPA query from an entity manager to access an EclipseLink persistence provider getDatabaseQuery method to set a new SQLCall.

***Call***

((org.eclipse.persistence.jpa.JpaQuery)query).

getDatabaseQuery().setCall(new SQLCall("..."));

For more information, see [Call Queries](http://wiki.eclipse.org/Introduction_to_EclipseLink_Queries_%28ELUG%29#Call_Queries).

**Using Named Parameters in a Native Query**

Using EclipseLink, you can specify a named parameter in a native query using the EclipseLink # convention (see the [Specifying a Named Parameter with #](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-31) example).

Support for the EclipseLink # convention is helpful if you are already familiar with EclipseLink queries or if you are migrating EclipseLink queries to a JPA application.

***Specifying a Named Parameter with #***

Query queryEmployees = entityManager.createNativeQuery(

"SELECT \* FROM EMPLOYEE emp WHERE emp.fname LIKE #firstname");

queryEmployees.setParameter("firstName", "Joan");

Collection employees = queryEmployees.getResultList();

**Using JP QL Positional Parameters in a Native Query**

Using EclipseLink, you can specify positional parameters in a native query using the Java Persistence query language (JP QL) positional parameter convention ?n to specify a parameter by number. For more information on JP QL, see [What You May Need to Know About Querying with Java Persistence Query Language](http://wiki.eclipse.org/Developing_Applications_Using_EclipseLink_JPA_%28ELUG%29#What_You_May_Need_to_Know_About_Querying_with_Java_Persistence_Query_Language).

This example shows how to specify positional parameters using the ?n convention. In this example, the query string will be SELECT \* FROM EMPLOYEE WHERE F\_NAME LIKE "D%" AND L\_NAME LIKE "C%".

***Specifying Positional Parameters Using ?***

Query queryEmployees = entityManager.createNativeQuery(

"SELECT \* FROM EMPLOYEE WHERE F\_NAME LIKE ?1 AND L\_NAME LIKE ?2", Employee.class);

queryEmployees.setParameter(1, "D%");

queryEmployees.setParameter(2, "C%");

Collection employees = queryEmployees.getResultList();

You can easily re-use the same parameter in more than one place in the query, as the following example shows. In this example, the query string will be SELECT \* FROM EMPLOYEE WHERE F\_NAME LIKE "D%" AND L\_NAME LIKE "D%".

***Specifying Positional Parameters Using ?n***

Query queryEmployees = entityManager.createNativeQuery(

"SELECT \* FROM EMPLOYEE WHERE F\_NAME LIKE ?1 AND L\_NAME LIKE ?1", Employee.class);

queryEmployees.setParameter(1, "D%");

Collection employees = queryEmployees.getResultList();

**Using JDBC-Style Positional Parameters in a Native Query**

Using EclipseLink, you can specify positional parameters in a native query using the JDBC-style positional parameter ? convention.

This example shows how to specify positional parameters using the ? convention. Each occurrence of ? must be matched by a corresponding setParameter call. In this example, the query string will be SELECT \* FROM EMPLOYEE WHERE F\_NAME LIKE "D%" AND L\_NAME LIKE "C%".

***Specifying Positional Parameters with ?***

Query queryEmployees = entityManager.createNativeQuery(

"SELECT \* FROM EMPLOYEE WHERE F\_NAME LIKE ? AND L\_NAME LIKE ?", Employee.class);

queryEmployees.setParameter(1, "D%");

queryEmployees.setParameter(2, "C%");

Collection employees = queryEmployees.getResultList();

If you want to re-use the same parameter in more than one place in the query, you must repeat the same parameter, as this example shows. In this example, the query string will be SELECT \* FROM EMPLOYEE WHERE F\_NAME LIKE "D%" AND L\_NAME LIKE "D%".

***Re-Using Positional Parameters with ?***

Query queryEmployees = entityManager.createNativeQuery(

"SELECT \* FROM EMPLOYEE WHERE F\_NAME LIKE ? AND L\_NAME LIKE ?", Employee.class);

queryEmployees.setParameter(1, "D%");

queryEmployees.setParameter(2, "D%");

Collection employees = queryEmployees.getResultList();

**Using EclipseLink JPA Weaving**

Weaving is a technique of manipulating the byte-code of compiled Java classes. The EclipseLink JPA persistence provider uses weaving to enhance JPA entities for such things as lazy loading, change tracking, fetch groups, and internal optimizations.

Weaving can be performed either dynamically at runtime, when Entities are loaded, or statically at compile time by post-processing the Entity .class files. By default, EclipseLink uses dynamic weaving whenever possible. This includes inside an Java EE 5/6 application server and in Java SE when the [EclipseLink agent](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Dynamic_Weaving_for_JPA_Entities_Using_the_EclipseLink_Agent) is configured. Dynamic weaving is recommended as it is easy to configure and does not require any changes to a project's build process.

This section describes the following:

* [How to Configure Dynamic Weaving for JPA Entities Using the EclipseLink Agent](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Dynamic_Weaving_for_JPA_Entities_Using_the_EclipseLink_Agent)
* [How to Configure Static Weaving for JPA Entities](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Static_Weaving_for_JPA_Entities)
* [How to Disable Weaving Using EclipseLink Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Disable_Weaving_Using_EclipseLink_Persistence_Unit_Propertie)
* [What You May Need to Know About Weaving JPA Entities](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Weaving_JPA_Entities)

**How to Configure Dynamic Weaving for JPA Entities Using the EclipseLink Agent**

Use this option to weave applicable class files one at a time, as they are loaded at run time. Consider this option when the number of classes to weave is few or the time taken to weave the classes is short.

If the number of classes to weave is large or the time required to weave the classes is long, consider using static weaving. For more information, see [How to Configure Static Weaving for JPA Entities](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Static_Weaving_for_JPA_Entities).

**To Configure Dynamic Weaving for JPA Entities Using the EclipseLink Agent**

1. Modify your application JVM command line to include the following:

-javaagent:eclipselink.jar

1. Ensure that the eclipselink.jar is in your application classpath.
2. Package and deploy your application.  
   For more information, see [Packaging an EclipseLink JPA Application](http://wiki.eclipse.org/Packaging_and_Deploying_EclipseLink_JPA_Applications_%28ELUG%29#Packaging_an_EclipseLink_JPA_Application).

EclipseLink weaves applicable class files one at a time, as they are loaded at run time.

**How to Configure Static Weaving for JPA Entities**

Use this option to weave all applicable class files at build time so that you can deliver pre-woven class files. Consider this option to weave all applicable class files at build time so that you can deliver prewoven class files. By doing so, you can improve application performance by eliminating the runtime weaving step required by dynamic weaving (see[How to Configure Dynamic Weaving for JPA Entities Using the EclipseLink Agent](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Dynamic_Weaving_for_JPA_Entities_Using_the_EclipseLink_Agent)).

In addition, consider using this option to weave in Java environments where you cannot configure an agent.

Prior to weaving, your persistence unit should be set-up in a way that is understood by eclipselink. There are two basic configurations:

**1. A jar file - setup as specified in the JPA specification**

* classes stored at the base in directories based on their package structure
* a META-INF directory containing your persistence.xml. Note: Using the persistenceunitinfo setting below, you can avoid this requirement

e.g. mypersistenceunit.jar could contain

* *mypackage/MyEntity1.class*
* *mypackage/MyEntity2.class*
* *mypackage2/MyEntity3.class*
* *META-INF/persistence.xml*

**2. An exploded directory structure**

* classes stored at the base in directories based on their package structure
* a META-INF directory containing your persistence.xml. Note: Using the persistenceunitinfo setting below, you can avoid this requirement

e.g. If your base directory was c:/classes, the exploded directory structure would look as follows:

* *c:/classes/mypackage/MyEntity1.class*
* *c:/classes/mypackage/MyEntity2.class*
* *c:/classes/mypackage2/MyEntity3.class*
* *c:/classes/META-INF/persistence.xml*

**To Configure Static Weaving for JPA Entities**

1. Execute the static static weaver in one of the following ways:
   1. Use the weave Ant task as follows:
      * Configure the weave Ant task in your build script, as this example shows. The [EclipseLink weave Ant Task Attributes](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Table_19-31) table lists the attributes of this task.  
        ***EclipseLink weave Ant Task***
      * <target name="define.task" description="New task definition for EclipseLink static weaving"/>
      * <taskdef name="weave" classname="org.eclipse.persistence.tools.weaving.jpa.StaticWeaveAntTask"/>
      * </target>
      * <target name="weaving" description="perform weaving" depends="define.task">
      * <weave source="c:\myjar.jar"
      * target="c:\wovenmyjar.jar"
      * persistenceinfo="c:\myjar-containing-persistenceinfo.jar">
      * <classpath>
      * <pathelement path="c:\myjar-dependent.jar"/>
      * </classpath>
      * </weave>
      * </target>

***EclipseLink weave Ant Task Attributes***

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Default** | **Required or Optional** |
| source | Specifies the location of the Java source files to weave: either a directory or a JAR file.  If the persistence.xml file is not in a META-INF directory at this location, you must specify the location of thepersistence.xml using the persistenceinfo attribute. |  | Required |
| target | Specifies the output location: either a directory or a JAR file. |  | Required |
| persistenceinfo | Specifies the location of the persistence.xml file if it is not in the same location as the source. Note: persistence.xml should be put in a directory called META-INF at this location |  | Optional |
| log | Specifies a logging file. | See[Logging](http://wiki.eclipse.org/Introduction_to_EclipseLink_Sessions_%28ELUG%29#Logging). | Optional |
| loglevel | Specifies the amount and detail of log output.  Valid java.util.logging.Level values are the following:   * + - * OFF       * SEVERE       * WARNING       * INFO       * CONFIG       * FINE       * FINER       * FINEST   For more information, see [Logging](http://wiki.eclipse.org/Introduction_to_EclipseLink_Sessions_%28ELUG%29#Logging). | Level.OFF | Optional |

|  |
| --- |
| **Note:** If source and target point to the same location and, if the source is a directory (not a JAR file), EclipseLink will weave in place. If source and target point to different locations, or if the source is a JAR file (as the [EclipseLink weave Ant Task](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-37)example shows), EclipseLink cannot weave in place. |

* + - Configure the weave task with an appropriate <classpath> element, as the [EclipseLink weave Ant Task](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-37) example shows, so that EclipseLink can load all required source classes.
    - Execute the Ant task using the command line that this example shows.  
      In this example, the weave Ant task is in the build.xml file:  
      ***EclipseLink weave Ant Task Command Line***
    - ant -lib C:\eclipselink.jar -f build.xml weave

|  |
| --- |
| **Note:** You must specify the eclipselink.jar file (the JAR that contains the EclipseLink weave Ant task) using the Ant command line -lib option instead of using the taskdef attribute classpath. |

* 1. Use the command line as follows:

java org.eclipse.persistence.tools.weaving.jpa.StaticWeave [arguments] <source> <target>

The following example shows how to use the StaticWeave class on Windows systems. The [EclipseLink StaticWeave Class Command Line Arguments](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Table_19-32) table lists the arguments of this class.  
***Executing StaticWeave on the Command Line***

java org.eclipse.persistence.tools.weaving.jpa.StaticWeave -persistenceinfo c:\myjar-containing-persistencexml.jar

-classpath c:\classpath1;c:\classpath2 c:\myjar-source.jar c:\myjar-target.jar

***EclipseLink StaticWeave Class Command Line Arguments***

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Description** | **Default** | **Required or Optional** |
| -persistenceinfo | Specifies the location of the persistence.xml file if it is not at the same location as the source (see -classpath) Note: EclipseLink will look in a META-INF directory at that location for persistence.xml. |  | Optional |
| -classpath | Specifies the location of the Java source files to weave: either a directory or a JAR file. For Windows systems, use delimiter " ; ", and for Unix, use delimiter " : ".  If the persistence.xml file is not in this location, you must specify the location of the persistence.xml using the -persistenceinfo attribute. |  | Required |
| -log | Specifies a logging file. | See[Logging](http://wiki.eclipse.org/Introduction_to_EclipseLink_Sessions_%28ELUG%29#Logging). | Optional |
| -loglevel | Specifies the amount and detail of log output.  Valid java.util.logging.Level values are as follows:   * + - OFF     - SEVERE     - WARNING     - INFO     - CONFIG     - FINE     - FINER     - FINEST   For more information, see [Logging](http://wiki.eclipse.org/Introduction_to_EclipseLink_Sessions_%28ELUG%29#Logging). | Level.OFF | Optional |
| <source> | Specifies the location of the Java source files to weave: either a directory or a JAR file.  If the persistence.xml file is not in this location, you must specify the location of the persistence.xml using the [-persistenceinfo](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#-persistenceinfo) attribute. |  | Required |
| <target> | Specifies the output location: either a directory or a JAR file. |  | Required |

|  |
| --- |
| **Note:** If <source> and <target> point to the same location and if the <source> is a directory (not a JAR file), EclipseLink will weave in place. If <source> and <target> point to different locations, or if the source is a JAR file (as the [Executing StaticWeave on the Command Line](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-39) example shows), EclipseLink cannot weave in place. |

1. Configure your persistence.xml file with a [eclipselink.weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.weaving) extension set to static, as this example shows:  
   ***Setting eclipselink.weaving in the persistence.xml File***
2. <persistence>
3. <persistence-unit name="HumanResources">
4. <class>com.acme.Employee</class>
5. ...
6. <properties>
7. <property
8. name="eclipselink.weaving"
9. value="static"
10. >
11. </properties>
12. </persistence-unit>
13. </persistence>

For more information, see the [EclipseLink JPA Persistence Unit Properties for Customization and Validation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Table_19-16) table.

1. Package and deploy your application.  
   For more information, see [Packaging and Deploying EclipseLink JPA Applications](http://wiki.eclipse.org/Packaging_and_Deploying_EclipseLink_JPA_Applications_%28ELUG%29#Packaging_and_Deploying_EclipseLink_JPA_Applications).

**How to Disable Weaving Using EclipseLink Persistence Unit Properties**

To disable weaving, you use persistence unit properties.

**To Disable Weaving Using EclipseLink Persistence Unit Properties**

1. Configure your persistence.xml file with one or more of the following properties set to false:
   * [eclipselink.weaving](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.weaving) – disables all weaving;
   * [eclipselink.weaving.lazy](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.weaving.lazy) – disables weaving for lazy loading (indirection);
   * [eclipselink.weaving.changetracking](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.weaving.changetracking) – disables weaving for change tracking;
   * [eclipselink.weaving.fetchgroups](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.weaving.fetchgroups) – disables weaving for fetch groups.
   * [eclipselink.weaving.internal](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.weaving.internal) – disables weaving for internal optimization.
   * [eclipselink.weaving.eager](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.weaving.eager) – disables weaving for indirection on eager relationships.

This example shows how to disable weaving for change tracking only.   
***Disabling Weaving for Change Tracking in the persistence.xml File***

<persistence>

<persistence-unit name="HumanResources">

<class>com.acme.Employee</class>

...

<properties>

<property

name="eclipselink.weaving.changetracking"

value="false"

/>

</properties>

</persistence-unit>

</persistence>

The following example shows how to disable all weaving: in this example, EclipseLink does not weave for lazy loading (indirection), change tracking, or internal optimization.

***Disabling All Weaving in the persistence.xml File***

<persistence>

<persistence-unit name="HumanResources">

<class>com.acme.Employee</class>

...

<properties>

<property

name="eclipselink.weaving"

value="false"

/>

</properties>

</persistence-unit>

</persistence>

For more information, see the [EclipseLink JPA Persistence Unit Properties for Customization and Validation](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Table_19-16) table.

1. Package and and deploy your application. For more information, see [Packaging and Deploying EclipseLink JPA Applications](http://wiki.eclipse.org/Packaging_and_Deploying_EclipseLink_JPA_Applications_%28ELUG%29#Packaging_and_Deploying_EclipseLink_JPA_Applications).

**What You May Need to Know About Weaving JPA Entities**

The EclipseLink JPA persistence provider uses weaving to enable the following features for JPA entities:

* lazy loading (indirection): see [How to Configure Lazy Loading](http://wiki.eclipse.org/Developing_Applications_Using_EclipseLink_JPA_%28ELUG%29#How_to_Configure_Lazy_Loading);
* change tracking: see [How to Configure Change Tracking](http://wiki.eclipse.org/Developing_Applications_Using_EclipseLink_JPA_%28ELUG%29#How_to_Configure_Change_Tracking);
* fetch groups: see [How to Configure Fetch Groups](http://wiki.eclipse.org/Developing_Applications_Using_EclipseLink_JPA_%28ELUG%29#How_to_Configure_Fetch_Groups);
* internal optimizations: see [Optimizing the EclipseLink Application](http://wiki.eclipse.org/Optimizing_the_EclipseLink_Application_%28ELUG%29).

EclipseLink weaves all the JPA entities in a given persistence unit. That is the following:

* all classes you list in persistence.xml file;
* if element <exclude-unlisted-classes> is false, or deployed in Java EE, all classes in the JAR file containing the persistence.xml file;
* all classes you list in the orm.xml file.

For more information, see [What You May Need to Know About Weaving and Java EE Application Servers](http://wiki.eclipse.org/Introduction_to_EclipseLink_Application_Development_%28ELUG%29#What_You_May_Need_to_Know_About_Weaving_and_Java_EE_Application_Servers).

**What You May Need to Know About EclipseLink JPA Lazy Loading**

JPA specifies that lazy loading is a hint to the persistence provider that data should be fetched lazily when it is first accessed, if possible.

If you are developing your application in a Java EE environment, you only have to set fetch to javax.persistence.FetchType.LAZY, and EclipseLink persistence provider will supply all the necessary functionality.

When using a [one-to-one](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40OneToOne) or [many-to-one](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40ManyToOne) mapping in a Java SE environment, to configure EclipseLink JPA to perform lazy loading when the fetch attribute is set toFetchType.LAZY, configure either dynamic or static weaving.

When using a [one-to-one](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40OneToOne) or [many-to-one](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40ManyToOne) mapping in a Java SE environment that does not permit the use of -javaagent on the JVM command line, to configure EclipseLink JPA to perform lazy loading when annotation attribute fetch is set to javax.persistence.FetchType.LAZY, you can use static weaving.

The [EclipseLink JPA Support for Lazy Loading by Mapping Type](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Table_19-33) table lists EclipseLink JPA support for lazy loading by mapping type.

For more information, see the following:

* [How to Configure Dynamic Weaving for JPA Entities Using the EclipseLink Agent](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Dynamic_Weaving_for_JPA_Entities_Using_the_EclipseLink_Agent)
* [How to Configure Static Weaving for JPA Entities](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Static_Weaving_for_JPA_Entities)
* [How to Disable Weaving Using EclipseLink Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Disable_Weaving_Using_EclipseLink_Persistence_Unit_Properties)
* [What You May Need to Know About Weaving JPA Entities](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_Weaving_JPA_Entities)
* [Configuring Indirection (Lazy Loading)](http://wiki.eclipse.org/Configuring_a_Mapping_%28ELUG%29#Configuring_Indirection_.28Lazy_Loading.29)

***EclipseLink JPA Support for Lazy Loading by Mapping Type***

|  |  |  |
| --- | --- | --- |
| **Mapping** | **Java EE 1** | **Java SE** |
| [many-to-many](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40ManyToMany) | EclipseLink JPA performs lazy loading when the fetch attribute is set tojavax.persistence.FetchType.LAZY (default). | EclipseLink JPA performs lazy loading when the fetch attribute is set tojavax.persistence.FetchType.LAZY (default). |
| [one-to-many](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40OneToMany) | EclipseLink JPA performs lazy loading when the fetch attribute is set tojavax.persistence.FetchType.LAZY (default). | EclipseLink JPA performs lazy loading when the fetch attribute is set tojavax.persistence.FetchType.LAZY (default). |
| [one-to-one](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40OneToOne) | EclipseLink JPA performs lazy loading when the fetch attribute is set tojavax.persistence.FetchType.LAZY. | By default, EclipseLink JPA ignores the fetch attribute and default javax.persistence.FetchType.EAGER applies.  To configure EclipseLink JPA to perform lazy loading when the fetch attribute set toFetchType.LAZY, consider one of the following:   * [How to Configure Dynamic Weaving for JPA Entities Using the EclipseLink Agent](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Dynamic_Weaving_for_JPA_Entities_Using_the_EclipseLink_Agent) * [How to Configure Static Weaving for JPA Entities](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Static_Weaving_for_JPA_Entities) |
| [many-to-one](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40ManyToOne) | EclipseLink JPA performs lazy loading when the fetch attribute is set tojavax.persistence.FetchType.LAZY. | By default, EclipseLink JPA ignores the fetch attribute and defaultjavax.persistence.FetchType.EAGER applies.  To configure EclipseLink JPA to perform lazy loading when the fetch attribute set toFetchType.LAZY, configure one of the following:   * [How to Configure Dynamic Weaving for JPA Entities Using the EclipseLink Agent](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Dynamic_Weaving_for_JPA_Entities_Using_the_EclipseLink_Agent) * [How to Configure Static Weaving for JPA Entities](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Static_Weaving_for_JPA_Entities) |
| [basic](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Basic) | EclipseLink JPA performs lazy loading when the fetch attribute is set tojavax.persistence.FetchType.LAZY. | By default, EclipseLink JPA ignores the fetch attribute and default javax.persistence.FetchType.EAGER applies.  To configure EclipseLink JPA to perform lazy loading when the fetch attribute set toFetchType.LAZY, consider one of the following:   * [How to Configure Dynamic Weaving for JPA Entities Using the EclipseLink Agent](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Dynamic_Weaving_for_JPA_Entities_Using_the_EclipseLink_Agent) * [How to Configure Static Weaving for JPA Entities](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#How_to_Configure_Static_Weaving_for_JPA_Entities) |

1Fully supported in any container that implements the appropriate container contracts in the EJB 3.0 specification.

**What You May Need to Know About Overriding Annotations in JPA**

In JPA, you override any annotation with XML in your object relational mapping files (see [Overriding Annotations with XML](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Overriding_Annotations_with_XML)).

In EclipseLink JPA, you either use JPA processing, or you specify the sessions.xml file resulting in creation of the project.xml file. For more information, see [What You May Need to Know About EclipseLink JPA Overriding Mechanisms](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_You_May_Need_to_Know_About_EclipseLink_JPA_Overriding_Mechanisms).

**Overriding Annotations with eclipselink-orm.xml File**

The eclipselink-orm.xml file is the EclipseLink native metadata XML file. It can be used to override the JPA configurations defined in the JPA orm.xml file. This exclipselink-orm.xml file provides access to the the advanced features provided by the EclipseLink XSD (<http://www.eclipse.org/eclipselink/xsds/eclipselink_orm_1_0.xsd>).

See [this page](http://wiki.eclipse.org/EclipseLink/Examples/JPA/EclipseLink-ORM.XML) for more information and [examples](http://wiki.eclipse.org/EclipseLink/Examples/JPA/EmployeeXML).

**Overriding Annotations with XML**

In JPA, you can use XML mapping metadata on its own, or in combination with annotation metadata, or you can use it to override the annotation metadata.

If you choose to include one or more mapping XML files in your persistence unit, each file must conform and be valid against the orm\_1\_0.xsd schema located at<http://java.sun.com/xml/ns/persistence/orm_1_0.xsd>. This schema defines a namespace called <http://java.sun.com/xml/ns/persistence/orm> that includes all of the ORM elements that you can use in your mapping file.

All object relational XML metadata is contained within the entity-mappings root element of the mapping file. The subelements of entity-mappings can be categorized into four main scoping and functional groups: persistence unit defaults, mapping file defaults, queries and generators, and managed classes and mappings. There is also a special setting that determines whether annotations should be considered in the metadata for the persistence unit (see [Disabling Annotations](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Disabling_Annotations)).

For more information and examples, see Section 10.1 "XML Overriding Rules" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220).

**Disabling Annotations**

JPA provides a mechanism that you can use to disable annotations. If you do not feel the need for annotations in your application, you can use the xml-mapping-metadata-complete and metadata-complete mapping file elements to disable any existing annotations. Setting this options causes the processor to completely ignore annotations.

When you specify the xml-mapping-metadata-complete element, all annotations in the persistence unit will be ignored, and only mapping files in the persistence unit will be considered as the total set of provided metadata. Only entities (see Section 8.1 "Entity" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220)), mapped superclasses (see [@MappedSuperclass](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40MappedSuperclass)), and embedded objects (see [@Embedded](http://wiki.eclipse.org/Introduction_to_EclipseLink_JPA_%28ELUG%29#.40Embedded)) that have entries in a mapping file will be added to the persistence unit. The xml-mapping-metadata-complete element has to be in only one of the mapping files in the persistence unit. You specify it as an empty subelement of the persistence-unit-metadata element, as this example shows:

***Disabling Annotations for the Persistence Unit in the Mapping File***

<entity-mappings>

<persistence-unit-metadata>

<xml-mapping-metadata-complete/>

</persistence-unit-metadata>

...

</entity-mappings>

You can also use the metadata-complete attribute of the entity, mapped-superclass, and embeddable elements. If you specify this attribute, all annotations on the specified class and on any fields or properties in the class will be ignored – only metadata in the mapping file will be considered as the set of metadata for the class. However, even though the annotations are ignored, the default mapping still applies, so any fields or properties that should not be mapped must still be marked as transient in the XML file, as this example demonstrates.

***Disabling Annotations for a Managed Class in the Mapping File***

@Entity

**public** **class** Employee {

@Id

**private** **int** id;

@Column(name="EMP\_NAME")

**private** String name;

@Column(name="SALARY")

**private** **long** salary;

...

}

**<entity-mappings>**

...

**<entity** class="mypackage.Employee" '''metadata-complete="true"'''**>**

**<attributes>**

**<id** name="id"**/>**

**</attributes>**

**</entity>**

...

**</entity-mappings>**

In the preceding example, the entity mappings in the annotated class are disabled by the metadata-complete attribute, and because the fields are not mapped in the mapping file, the default mapping values will be used. The name and salary fields will be mapped to the NAME and SALARY columns, respectively.

For more information, see Section 10.1 "XML Overriding Rules" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220).

**Advantages and Disadvantages of Using Annotations**

Metadata annotations are relatively simple to use and understand. They provide in-line metadata located with the code that this metadata is describing – you do not need to replicate the source code context of where the metadata applies.

On the other hand, annotations unnecessarily couple the metadata to the code. Thus, changes to metadata require changing the source code.

**Advantages and Disadvantages of Using XML**

The following are the advantages of using XML:

* no coupling between the metadata and the source code;
* compliance with the existing, pre-EJB 3.0 development process;
* support in IDEs and source control systems.

The main disadvantages of mapping with XML are the complexity and the need for replication of the code context.

**What You May Need to Know About EclipseLink JPA Overriding Mechanisms**

EclipseLink JPA provides a set of persistence unit properties (see [What you May Need to Know About Using EclipseLink JPA Persistence Unit Properties](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#What_you_May_Need_to_Know_About_Using_EclipseLink_JPA_Persistence_Unit_Properties)) that you can specify in your persistence.xml file. The persistence unit properties always override the corresponding annotations' attributes.

Similar to EclipseLink annotations, properties expose some features of EclipseLink that are currently not available through the use of JPA metadata.

|  |
| --- |
| **Note:** If multiple instances of the same property are set, then EclipseLink will use the values from the last entry in the list. However, the properties Map provided in the createEntityManagerFactory method will always have precedence. |

You can also specify the persistence information in the EclipseLink session configuration file – sessions.xml (see [Session Configuration and the sessions.xml File](http://wiki.eclipse.org/Introduction_to_EclipseLink_Sessions_%28ELUG%29)). By setting the [eclipselink.sessions-xml](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.sessions-xml) persistence unit property you enable EclipseLink to replace all class annotations and object relational mappings that you defined in thepersistence.xml file, as well as mapping files (if present). Through the sessions.xml file the eclipselink.sessions-xml property lets you provide session-level configurations that are not supported by persistence unit properties (for example, cache coordination).

|  |
| --- |
| **Note:** You can use the eclipselink.sessions-xml property as an alternative to annotations and deployment XML. |

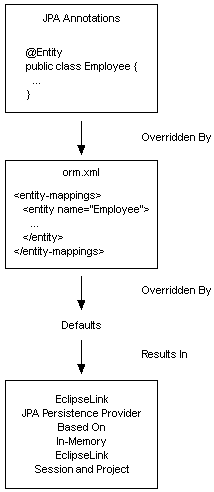
For more information on creating and configuring the sessions.xml file, see the following:

* [Acquiring a Session at Run Time with the Session Manager](http://wiki.eclipse.org/Introduction_to_EclipseLink_Sessions_%28ELUG%29)
* [Building and Using the Persistence Layer](http://wiki.eclipse.org/Introduction_to_EclipseLink_Application_Development_%28ELUG%29)
* [Loading project.xml or sessions.xml Files](http://wiki.eclipse.org/Integrating_EclipseLink_with_an_Application_Server_%28ELUG%29)
* [Development Environment](http://wiki.eclipse.org/Introduction_to_EclipseLink_Development_Tools_%28ELUG%29)
* [Introduction to the EclipseLink Deployment File Creation](http://wiki.eclipse.org/Creating_EclipseLink_Files_for_Deployment_%28ELUG%29)
* [Packaging an EclipseLink Application](http://wiki.eclipse.org/Packaging_a_EclipseLink_Application_%28ELUG%29)
* [EclipseLink Sessions XML File](http://wiki.eclipse.org/Introduction_to_XML_Projects_%28ELUG%29)
* [Introduction to the Session Creation](http://wiki.eclipse.org/Creating_a_Session_%28ELUG%29)
* [Creating a Sessions Configuration](http://wiki.eclipse.org/Creating_a_Session_%28ELUG%29)
* [Configuring a Sessions Configuration](http://wiki.eclipse.org/Creating_a_Session_%28ELUG%29)
* [Introduction to Session Acquisition](http://wiki.eclipse.org/Acquiring_and_Using_Sessions_at_Run_Time_%28ELUG%29)

In summary, EclipseLink JPA possesses an overriding mechanism that you can use in the following ways:

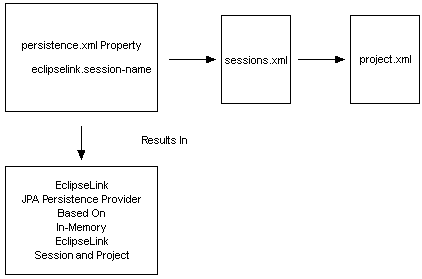
* You can combine the use of JPA annotations, object relational mapping files (such as orm.xml) and persistence unit properties.In this case, EclipseLink persistence provider builds metadata starting with applying defaults, then JPA annotations, and then overrides that with elements of the object relational mapping file. This results in creation of an in-memory EclipseLink session and project. Then EclipseLink persistence provider applies persistence unit properties specified in persistence.xml file, as the following illustration shows.

|  |
| --- |
| **Note:** The [eclipselink.sessions-xml](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.sessions-xml) property represents a special case discussed further. |

***Combining the Use of Annotations, orm.xml File and Persistence Unit Properties***  
[](http://wiki.eclipse.org/Image:Override_spec.gif)

* You can use the [eclipselink.sessions-xml](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.sessions-xml) persistence unit property. This defines a sessions.xml file, which references a project.xml file. In this case, EclipseLink persistence provider builds an in-memory EclipseLink session and project based on this metadata, as the following illustration shows. You can acquire a persistence manager and use it as per the JPA specification, having defined all entities and so on using only EclipseLink sessions.xml (see [Creating Session Metadata](http://wiki.eclipse.org/Introduction_to_EclipseLink_Application_Development_%28ELUG%29#Creating_Session_Metadata)) andproject.xml files that you created Workbench (see [Creating the project.xml File with Workbench|Creating the project.xml File with Workbench](http://wiki.eclipse.org/Creating_EclipseLink_Files_for_Deployment_%28ELUG%29)).

***Using eclipselink.sessions-xml Persistence Unit Property***

[](http://wiki.eclipse.org/Image:Override_tl.gif)

|  |
| --- |
| **Note:** You cannot combine the use of JPA annotations, object relational mapping files, persistence unit properties and the[eclipselink.sessions-xml](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#eclipselink.sessions-xml) persistence unit property. |

**What You May Need to Know About Using EclipseLink JPA Persistence Unit Properties**

A persistence unit configures various details that are required when you acquire an entity manager. You specify a persistence unit by name when you acquire an entity manager factory.

You configure persistence units in JPA persistence descriptor file persistence.xml.

In this file, you can specify the vendor extensions that this reference describes by using a <properties> element. The [Configuring a Vendor Extension in the persistence.xml File (Java EE)](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-45) example shows how to set an EclipseLink JPA persistence unit extension in a persistence.xml file for a Java EE application. The [Configuring a Vendor Extension in the persistence.xml File (Java SE)](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-46) example shows how to do the same for a Java SE application.

|  |
| --- |
| **Note:** that EclipseLink does not provide a mechanism for encrypting the password in the persistence.xml file. (see [Avoiding Cleartext Passwords](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Avoiding_Cleartext_Passwords)) |

***Configuring a Vendor Extension in the persistence.xml File (Java EE)***

**<persistence-unit** name="default" transaction-type="JTA"**>**

**<provider>**

org.eclipse.persistence.jpa.PersistenceProvider

**</provider>**

**<jta-data-source>**

jdbc/MyDataSource

**</jta-data-source>**

**<properties>**

**<property** name="eclipselink.logging.level" value="INFO"**/>**

**</properties>**

**</persistence-unit>**

***Configuring a Vendor Extension in the persistence.xml File (Java SE)***

**<persistence-unit** name="default" transaction-type="RESOURCE\_LOCAL"**>**

**<provider>**

org.eclipse.persistence.jpa.PersistenceProvider

**</provider>**

**<exclude-unlisted-classes>**false**</exclude-unlisted-classes>**

**<properties>**

**<property** name="eclipselink.logging.level" value="INFO"**/>**

**<property** name="eclipselink.jdbc.driver" value="oracle.jdbc.OracleDriver"**/>**

**<property** name="eclipselink.jdbc.url" value="jdbc:oracle:thin:@myhost:1521:MYSID"**/>**

**<property** name="eclipselink.jdbc.password" value="tiger"**/>**

**<property** name="eclipselink.jdbc.user" value="scott"**/>**

**</properties>**

**</persistence-unit>**

Alternatively, you can set a vendor extensions in the Map of properties you pass into a call to javax.persistence.Persistence method createEntityManagerFactory, as the[Configuring a Vendor Extension when Creating an EntityManagerFactory](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-47) example shows. You can override extensions set in the persistence.xml file in this way. When you set an extension in a map of properties, you can set the value using the public static final field in the appropriate configuration class in org.eclipse.persistence.config, including the following:

* CacheType
* TargetDatabase
* TargetServer
* PersistenceUnitProperties

The following example shows how to set the value of extension eclipselink.cache.type.default using the CacheType configuration class.

***Configuring a Vendor Extension when Creating an EntityManagerFactory***

import org.eclipse.persistence.config.CacheType;

Map properties = new HashMap();

properties.put(PersistenceUnitProperties.CACHE\_TYPE\_DEFAULT, CacheType.Full);

EntityManagerFactory emf = Persistence.createEntityManagerFactory("default", properties);

You may specify any EclipseLink JPA extension in a persistence.xml file and you may pass any EclipseLink JPA extension into Persistence methodcreateEntityManagerFactory.

Currently, no EclipseLink JPA extensions are applicable to EntityManagerFactory method createEntityManager.

For more information, see the following:

* Section 2.1 "Requirements on the Entity Class" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220);
* Chapter 5 "Entity Managers and Persistence Contexts" of the [JPA Specification](http://jcp.org/en/jsr/detail?id=220).

**Allowing Zero Value Primary Keys**

By default, EclipseLink interprets zero as *null* for primitive types that cannot be null (such as int and long) causing zero to be an invalid value for primary keys. You can modify this setting by using the **allow-zero-id property** in the persistence.xml file. Valid values are:

* **true** – EclipseLink interprets zero values as *zero*. This permits primary keys to use a value of zero.
* **false** (default) – EclipseLink interprets zero as *null*.

**Avoiding Cleartext Passwords**

EclipseLink does not support storing encrypted passwords in the persistence.xml file. For a Java EE application, you do not need to specify your password in thepersistence.xml file. Instead, you can specify a data-source, as shown in the [Java EE example.](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-45) This datasource is specified on the application server, and can encrypt the your password with its own mechanism.

For a Java SE application, you should avoid putting the password in the persistence.xml file. Instead, add the password to the properties, as shown in the [Setting the password in code example](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_(ELUG)#Example_19-48). Setting the password in code allows for the password to be retrieved from any location, including a protected source.

***Setting the password in code***

Map properties = **new** HashMap();

properties.put(PersistenceUnitProperties.JDBC\_PASSWORD, "tiger");

EntityManagerFactory emf = Persistence.createEntityManagerFactory("default", properties);