Oracle® Fusion Middleware

Oracle TopLink Concepts

11*g* Release 1 (11.1.1)

E26045-01

November 2011

This document provides conceptual information about the components and technologies that comprise Oracle TopLink.

Oracle Fusion Middleware Oracle TopLink Concepts, 11*g* Release 1 (11.1.1)

E26045-01

Primary Author: Thomas Pfaeffle

Contributing Author: Doug Clarke, Blaise Doughan, Peter Krogh, Gordon Yorke

Contents

Preface

Oracle TopLink links object-oriented programs with relational data structures. Using TopLink, you can build high-performance applications that store persistent object-oriented data in a relational database. TopLink successfully transforms object-oriented data into either relational data or XML documents. Using TopLink, you can integrate persistence and object-transformation into your application, while staying focused on your primary domain problem by taking advantage of an efficient, flexible, and field-proven solution.

Audience

This document is intended for application developers and administrators who want to know more about the concepts behind TopLink and its features.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle TopLink documentation set:

- *Solution Guide for Oracle TopLink*
- *Oracle Fusion Middleware Java API Reference for EclipseLink*
- *EclipseLink Documentation Center* at `http://wiki.eclipse.org/EclipseLink/UserGuide/`
- "Oracle TopLink" in *Oracle Fusion Middleware Release Notes for Linux x86*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 General Concepts

This chapter contains the following sections:

1.1 What is TopLink?

Oracle TopLink links object-oriented programs with relational data structures. Using TopLink, you can build high-performance applications that store persistent object-oriented data in a relational database. TopLink successfully transforms object-oriented data into either relational data or XML documents. You can integrate persistence and object-transformation into your application, while staying focused on your primary domain problem by taking advantage of an efficient, flexible, and field-proven solution.

TopLink includes EclipseLink. EclipseLink includes the open source implementation of the JPA specification, plus extensions beyond what is defined in the Java Persistence API (JPA) and Java API for XML Binding (JAXB) specifications. These extensions include persistence unit properties, query hints, annotations, TopLink's own XML metadata, and custom API. For a more detailed description of the contents of TopLink, see Section 1.4, "TopLink Components."

TopLink can be used with a wide range of Java Enterprise Edition (Java EE) and Java application architectures (see Section 1.5, "TopLink Application Architectures"). Use TopLink to design, implement, deploy, and optimize an advanced, object-persistence and object-transformation layer that supports a variety of data sources and formats, including the following:

- Relational—for transactional persistence of Java objects to a relational database accessed using Java Database Connectivity (JDBC) drivers.
- Object-Relational Data Type—for transactional persistence of Java objects to special purpose structured data source representations optimized for storage in object-relational data type databases such as Oracle Database.

- Object-XML—for nontransactional, nonpersistent (in-memory) conversion between Java objects and XML Schema Document (XSD)-based XML documents using JAXB.

TopLink supports Enterprise JavaBeans (EJB) 3.0 in Java EE and Java SE environments including integration with a variety of application servers, such as Oracle WebLogic Server, Glassfish Server, and IBM WebSphere application server.

The extensive suite of development tools that TopLink provides, including Oracle JDeveloper TopLink Editor, lets you quickly capture and define object-to-data source and object-to-data representation mappings in a flexible, efficient metadata format.

The TopLink runtime lets your application exploit this mapping metadata with a simple session facade that provides in-depth support for data access, queries, transactions (both with and without an external transaction controller), and caching.

For more information about TopLink, see Section 1.2, "TopLink Key Features."

1.1.1 What Is the Object-Persistence Impedance Mismatch?

Java-to-data source integration is a widely underestimated problem when creating enterprise Java applications. This complex problem involves more than simply reading from and writing to a data source. The data source elements include tables, rows, columns, and primary and foreign keys. The Java and Java EE programming languages include entity classes (regular Java classes or EJB entity beans), business rules, complex relationships, and inheritance. In a nonrelational data source, you must match your Java entities with XML elements and schemas.

Successful solution requires bridging these different technologies, and solving the object-persistence impedance mismatch—a challenging and resource-intensive problem. To solve this problem, you must resolve the following issues between Java EE and data source elements:

- Fundamentally different technologies
- Different skill sets
- Different staff and ownership for each of the technologies
- Different modeling and design principles

As an application developer, you need a product that lets you integrate Java applications with any data source, without compromising ideal application design or data integrity. In addition, as a Java developer, you need the ability to store (that is, persist) and retrieve business domain objects using a relational database or a nonrelational data source as a repository.

1.1.2 The TopLink Solution

TopLink addresses the disparity between Java objects and data sources. TopLink is a persistence framework that manages relational, object-relational data type, and XML mappings in a seamless manner. This lets you rapidly build applications that combine the best aspects of object technology and the specific data source. TopLink lets you do the following:

- Persist Java objects to virtually any relational database supported by a JDBC-compliant relational database
- Perform in-memory conversions between Java objects and XML Schema (XSD) based XML documents using JAXB

- Map any object model to any relational or nonrelational schema, using Oracle JDeveloper TopLink editor
- Generate TopLink DBWS service descriptors and accompanying files, using JDeveloper DBWSBuilder
- Use TopLink successfully, even if you are unfamiliar with SQL or JDBC, because TopLink offers a clear, object-oriented view of data sources

1.2 TopLink Key Features

TopLink provides an extensive set of features. You can use these features to rapidly build high-performance enterprise applications that are scalable and maintainable.

Some of the primary features of TopLink are the following:

- Nonintrusive, flexible, metadata-based architecture
- Architectural flexibility: Plain Old Java Objects (POJO), as well as JPA, JAXB, Service Data Objects (SDO), and Web services provided by EclipseLink
- Advanced mapping support and flexibility: relational, object-relational data type, and XML
- Optimized for highly scalable performance and concurrency with extensive performance tuning options
- Comprehensive object caching support including cluster integration for some application servers (such as, for example, Oracle Application Server)
- Extensive query capability including: TopLink Expressions framework, Java Persistence Query Language (JPQL), Enterprise JavaBeans Query Language (EJB QL), and native SQL
- Just-in-time reading
- Object-level transaction support and integration with popular application servers and databases
- Optimistic and pessimistic locking options and locking policies
- Comprehensive visual design tools: Oracle JDeveloper TopLink Editor and Eclipse Dali

For additional information and downloads, see the TopLink home page:

`http://www.oracle.com/technology/products/ias/toplink/index.html`

1.3 TopLink Metadata

TopLink metadata is the bridge between the development of an application and its deployed run-time environment. You can capture the metadata using:

- JPA annotations in Java files, and the JPA-defined properties in the `persistence.xml`, `eclipselink-orm.xml`, and `orm.xml` files. Metadata is also captured by TopLink JPA annotations and TopLink property extensions in the `persistence.xml` file. The `eclipselink-orm.xml` file can also be used to specify TopLink property extensions beyond the JPA specification.
- JAXB annotations in Java files and JAXB-defined properties in the `eclipselink-oxm.xml` file. The `eclipselink-oxm.xml` file can be used to define TopLink property extensions beyond the JAXB specification.
- Java and the EclipseLink API

The metadata lets you pass configuration information into the run-time environment. The run-time environment uses the information in conjunction with the persistent classes, such as Java objects, JPA entities, and the code written with the TopLink API, to complete the application. See Section 2.1.2, "Adding Metadata Using Annotations" for more information.

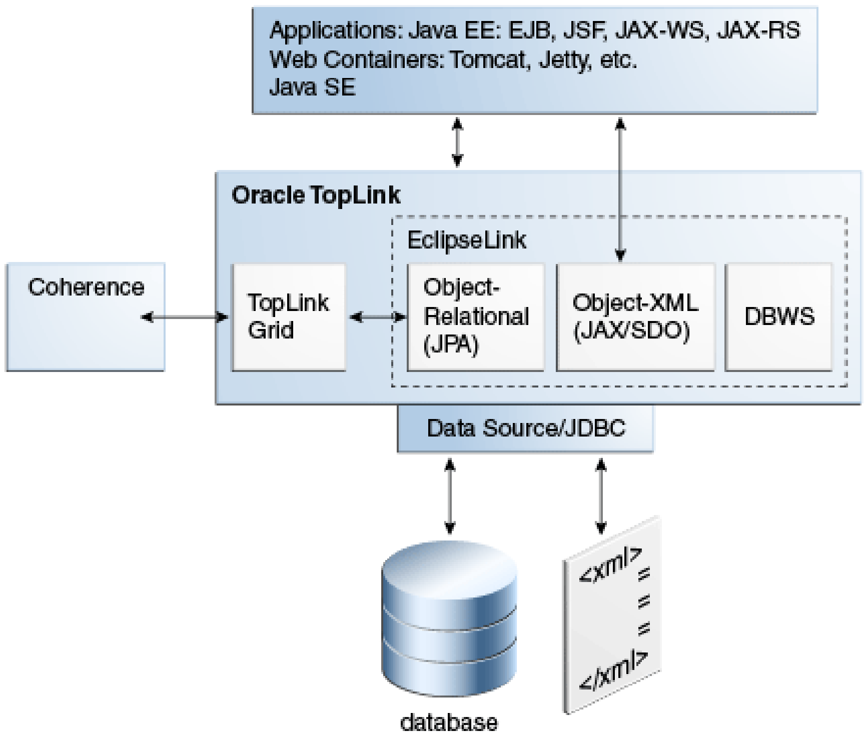
Mappings can be stored external to the application. This can be as simple as making the `eclipselink-orm.xml` or `eclipselink-oxm.xml` file with the additional mapping information available on a Web server as a file. It can also be more complex involving a server process that stores the mapping information and allows the information to be updated dynamically. For more information, see "EclipseLink/Examples/JPA/MetadataSource" in the EclipseLink documentation.

`http://wiki.eclipse.org/EclipseLink/Examples/JPA/MetadataSource`

1.4 TopLink Components

Figure 1–1 illustrates the components contained by Oracle TopLink. The following sections describe the components.

Figure 1–1 TopLink Components



1.4.1 EclipseLink Core

The EclipseLink Core provides the EclipseLink run-time component. Access to the run-time component can be obtained directly through the EclipseLink API. The run-time environment is not a separate or external process—it is embedded within the application. Application calls invoke EclipseLink to provide persistence behavior. This function allows for transactional and thread-safe access to shared database connections and cached objects. For more information, see Section 1.7, "TopLink/EclipseLink API."

1.4.2 Object-Relational (JPA 2.0)

JPA, part of the Java EE EJB 3.0 specification, greatly simplifies Java persistence. It 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. JPA works both inside a Java EE application server and outside an EJB container in a Java Standard Edition (Java SE) application. The main features included in the 2.0 JPA update are:

- Expanded object/relational mapping functionality
  - support for collections of embedded objects
  - multiple levels of embedded objects
  - ordered lists
  - combinations of access types
- A criteria query API
- Standardization of query "hints"
- Standardization of additional metadata to support DDL generation
- Support for validation

1.4.3 Object-XML (JAXB)

Object-XML, also known as *MOXy*, is a TopLink component that enables you to bind Java classes to XML schemas. Object-XML implements JAXB which allows you to provide mapping information through annotations and provide support for storing the mappings in XML format. The many advanced mappings which are available enable you to handle complex XML structures without having to mirror the schema in your Java class model.

The objects produced by the TopLink JAXB compiler are Java POJO models. They implement the necessary interfaces required by the JAXB specification. The JAXB runtime API can be used to marshal and unmarshal objects.

When using Object-XML as the JAXB provider, no metadata is required to convert your existing object model to XML. You can supply metadata (using annotations or XML) only when you need to fine-tune the XML representation of the model.

Using TopLink Object-XML, you can manipulate XML in the following ways:

- Generate a Java Model from an XML schema
- Specify the EclipseLink MOXy JAXB runtime
- Use JAXB to manipulate XML
- Generate an XML schema from a Java model

For more information on Object-XML and these use cases, see "Getting Started with MOXy" in the EclipseLink documentation:

`http://www.eclipse.org/eclipselink/moxy.php`

TopLink provides maximum flexibility with the ability to control how your object model is mapped to an XML schema. There are many advantages to having control over your own object model:

- The domain classes can be designed specifically for your application using the appropriate patterns and practices.

- XPath-based mapping. This prevents the need for having a 1-to-1 relationship between classes and XML schema types. For more information, see "Mapping Simple Values" in the EclipseLink documentation.

  `http://wiki.eclipse.org/EclipseLink/UserGuide/MOXy/Simple_Values/Single_Values/XMLDirectMapping`

- Can instantiate objects in a way that is appropriate to your application (that is, using the default constructor).
- Can control your own class path dependencies. Most JAXB implementations put vendor specific code in the generated classes that add class path dependencies to your application.

One of TopLink's key advantages is that the mapping information can be stored externally and does not require any changes to the Java classes or XML schema. This means that you may map your domain objects to more than one schema or if your schema changes you can simply update the mapping metadata instead of modifying your domain classes. This is also useful when mapping third-party classes, as you might not have access to the source to add annotations. You can also use annotations to provide the metadata.

1.4.3.1 TopLink SDO

The Service Data Objects (SDO) component provides the reference implementation of Service Data Objects version 2.1.1. The TopLink SDO implementation incorporates the reference implementation and provides additional features primarily used for converting Java objects to XML, and for building and using data object models that can be incorporated into service architectures.

TopLink SDO provides you with the following capabilities:

- use of the SDO APIs
- convert an XML Schema to SDO metadata
- customize your XSD for SDO usage
- use dynamic data objects
  - use dynamic data objects
  - use dynamic data objects to manipulate XML
- use of static data objects
  - run the SDO compiler—generate type safe data objects
  - use type safe data objects to manipulate XML

For more information, see "Getting Started with EclipseLink SDO" in the EclipseLink documentation:

`http://www.eclipse.org/eclipselink/moxy.php`

1.4.4 Database Web Services (DBWS)

TopLink Database Web Services (DBWS) enables simple and efficient access to relational database artifacts by using a Web service. It provides Java EE-compliant client-neutral access to the database without having to write Java code. TopLink DBWS extends TopLink's core capabilities while leveraging existing ORM and OXM components.

TopLink DBWS has two parts: a design-time component (DBWSBuilder) and a runtime provider component that takes a service descriptor (along with related deployment artifacts) and realizes it as a JAX-WS 2.0 Web service. The runtime provider uses EclipseLink to bridge between the database and the XML SOAP Messages used by Web service clients. For information on DBWS architecture, see Section 1.5.4, "TopLink Database Web Services."

1.4.5 TopLink Grid

Oracle TopLink enables you to scale out JPA applications using Oracle Coherence, and to write to the grid, treating the grid as the data source. TopLink Grid provides applications with a number of options on how they can scale, ranging from using Coherence as a distributed shared (L2) cache up to directing JPQL queries to Coherence for parallel execution across the grid to reduce database load. With TopLink Grid, you do not have to rewrite your applications to scale out.

TopLink Grid integrates the TopLink JPA implementation (EclipseLink) with Oracle Coherence and provides these development approaches:

- You can use the Coherence API with caches backed by TopLink Grid to access relational data with special cache loader and cache store interfaces which have been implemented for JPA.
- You can build applications using JPA and transparently use the power of the data grid for improved scalability and performance. In this JPA on the Grid approach, TopLink Grid provides a set of cache and query configuration options that allow you to control how TopLink JPA uses Coherence.

Note: TopLink Grid functionality is provided by the `toplink-grid.jar` file. This file is available only if you have also licensed Oracle Coherence.

1.5 TopLink Application Architectures

You can use TopLink in a variety of application architectures, including three- and two-tier architectures, with or without Java EE, to access a variety of data types on both relational and nonrelational data sources.

1.5.1 Three-Tier Architectures

The three-tier (or Java EE Web) application is one of the most common TopLink architectures. This architecture is characterized by a server-hosted environment in which the business logic, persistent entities, and the Oracle TopLink Foundation Library all exist in a single Java Virtual Machine (JVM).

The most common example of this architecture is a simple three-tier application in which the client browser accesses the application through servlets, JavaServer Pages (JSP) and HTML. The presentation layer communicates with TopLink through other Java classes in the same JVM, to provide the necessary persistence logic. This architecture supports multiple servers in a clustered environment, but there is no separation across JVMs from the presentation layer and the code that invokes the persistence logic against the persistent entities using TopLink.

1.5.1.1 EJB Session Bean Facade

A popular variation on the three-tier application involves wrapping the business logic, including the TopLink access, in EJB session beans. This architecture provides a scalable deployment and includes integration with transaction services from the host application server.

Communication from the presentation layer occurs through calls to the EJB session beans. This architecture separates the application into different tiers for the deployment. The session bean architecture can persist either Java objects or EJB entity beans

1.5.2 Java SE or Thick Client

A two-tier (or client/server) application is one in which the application accesses TopLink directly. Although less common than the other architectures discussed here, TopLink supports this architecture for smaller or embedded data processing applications.

1.5.3 Web Services

A Web services architecture is similar to the Web or session-bean architecture. However, in a Web services architecture you encapsulate business logic (the service) in a Web service instead of (or in addition to) using session beans. In a Web services architecture, clients communicate with your application using XML.

As in any architecture, you can use TopLink to persist objects to relational data sources. However, in a Web services architecture you can also use TopLink to map your object model to an XML schema for use with the Web service or as the Web service XML serializer.

1.5.4 TopLink Database Web Services

TopLink database Web services architecture is similar to the Web services architecture. However, in a TopLink database Web services architecture, you use TopLink to automatically generate Web services that expose database operations such as queries, DML statements, stored procedures, and stored functions. Using TopLink database Web services, you can provide Java EE-compliant, client-neutral access to a relational database without having to write Java code.

As in any Web services architecture, clients communicate with your application using SOAP (XML) messages. However, in a TopLink database Web services architecture you need only specify an XSD for persistent classes. Clients need only invoke the operations the TopLink database Web service exposes to create, read, update, and delete these persistent objects. TopLink database Web services return objects or row set data, depending on the type of operation.

1.6 Mappings

Mapping refers to relating an object or XML schema to a corresponding relational database table. TopLink can transform data between an object representation (such as a Java class) and a representation specific to a data source (such as a database table). This transformation is called mapping and it is the core of an TopLink project.

A mapping corresponds to a single data member of a domain object. It associates the object data member with its data source representation and defines the means of performing the two-way conversion between object and data source.

TopLink is a metadata-driven mapping engine with the mappings for different data sources provided through annotations, XML, or for advanced use cases, they can be written and augmented by using code.

For more information, see Section 1.6.2, "Object-Relational Data Type Mappings" and Section 1.6.3, "Object-XML Data Type Mappings."

1.6.1 Relational Mappings

Relational mappings transform any object data member type to a corresponding relational database (SQL) data source representation in any supported relational database. Relational mappings allow you to map an object model into a relational data model.

Relational mappings can also transform object data members that reference other domain objects that are stored in other tables in the database and are related through foreign keys.

1.6.2 Object-Relational Data Type Mappings

Object-relational data type mappings transform certain object data member types to structured data source representations optimized for storage in specialized object-relational databases such as Oracle Database. Object-relational data type mappings let you map an object model into an object-relational model. You can use only object-relational data type mappings with specialized object-relational databases optimized to support object-relational data type data source representations.

For more information, see "Object-Relational Data Type Mappings" in the EclipseLink documentation:

`http://wiki.eclipse.org/Object-Relational_Data_Type_Mappings_%28ELUG%29`

1.6.3 Object-XML Data Type Mappings

XML mappings transform object data members to the XML elements of an XML document whose structure is defined by an XML schema document (XSD). You can map the attributes of a Java object to a combination of XML simple and complex types using a wide variety of XML mapping types.

Classes are mapped to complex types, object relationships map to XML elements, and simple attributes map to text nodes and XML attributes. The real power is that when mapping an object attribute to an XML document, XPath statements are used to specify the location of the XML data.

TopLink stores XML mappings for each class in the class descriptor. TopLink uses the descriptor to instantiate objects mapped from an XML document and to store new or modified objects as an XML document.

TopLink provides XML mappings that are not defined in the JPA specification. Some of the Object-XML extensions are available through TopLink annotations, others require programmatic changes to the underlying metadata.

For more information on these mappings, see "Mapping Simple Values" in the EclipseLink documentation:

`http://wiki.eclipse.org/EclipseLink/UserGuide/MOXy/Simple_Values/Single_Values/XMLDirectMapping#Mapping_to_an_Attribute`

1.7 TopLink/EclipseLink API

The EclipseLink API component of TopLink provides the reference implementation for JPA 2.0 (JSR-317). The `org.eclipse.*` classes encapsulate the EclipseLink API and provide extensions beyond the specification. These extensions include EclipseLink-specific properties and annotations. For more information on the API, properties and extensions, see the *Oracle Fusion Middleware Java API Reference for EclipseLink*. See also "Using EclipseLink JPA Extensions" in the EclipseLink documentation:

`http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_%28ELUG%29`

The EclipseLink API component also includes the JPA interfaces in the `javax.persistence` package.

The older Native TopLink, "Classic" TopLink, and TopLink Essentials persistence products, which were represented by the `oracle.toplink.*` classes, have been removed from TopLink and replaced with the EclipseLink implementation.

2 Building Blocks of a TopLink Project

This chapter describes the items that can be used in a TopLink project.

This chapter contains the following sections:

- Section 2.1, "Building Blocks for Object-Relational Mapping"
- Section 2.2, "Building Blocks for Object-XML Mapping"

2.1 Building Blocks for Object-Relational Mapping

TopLink provides a complete, JPA 2.0-compliant JPA implementation. It provides complete compliance for all of the mandatory features, many of the optional features, and some additional features. The additional nonmandatory functionality includes object-level cache, distributed cache coordination, extensive performance tuning options, enhanced Oracle Database support, advanced mappings, optimistic and pessimistic locking options, extended annotations and query hints.

For more information, see "The EclipseLink JPA User's Guide" in the EclipseLink documentation:

`http://wiki.eclipse.org/EclipseLink/UserGuide/JPA`

The following sections describes many of these features.

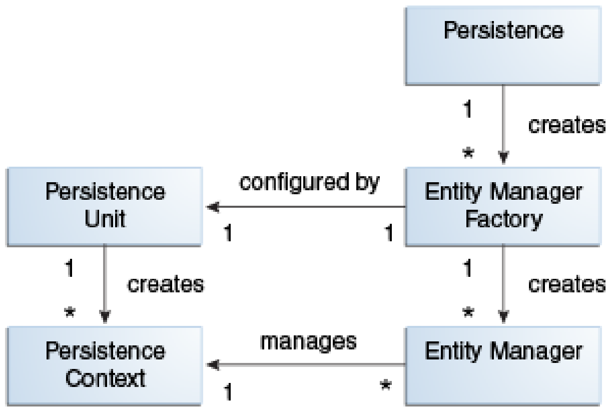
- Section 2.1.1, "Object-Relational Entity Architecture"
- Section 2.1.2, "Adding Metadata Using Annotations"
- Section 2.1.3, "Configuration Files"
- Section 2.1.4, "Data Sources"
- Section 2.1.5, "TopLink Caches"
- Section 2.1.6, "Database Queries"

2.1.1 Object-Relational Entity Architecture

The entity architecture is comprised of entities, persistence units, persistence contexts, entity manager factories, and entity managers. Figure 2–1 illustrates the relationships between these elements:

- Persistence creates one or more `EntityManagerFactory` objects
- Each `EntityManagerFactory` is configured by one persistence unit
- `EntityManagerFactory` creates one or more `EntityManager` objects
- One or more `EntityManagers` manage one `PersistenceContext`

Figure 2–1 Relationships Between Entity Architecture Elements



2.1.1.1 Entities

An entity is any application-defined object with the following characteristics:

- It can be made persistent.
- It has a persistent identity (a key that uniquely identifies an entity instance and distinguishes it from other instances of the same entity type. An entity has a persistent identity when there is a representation of it in a data store).
- It is transactional in a sense that a *persistence view* of an entity is transactional (an entity is created, updated and deleted within a transaction, and a transaction is required for the changes to be committed in the database). However, in-memory entities can be changed without the changes being persisted.
- It is *not* a primitive, a primitive wrapper, or built-in object. An entity is a fine-grained object that has a set of aggregated state that is typically stored in a single place (such as a row in a table), and have relationships to other entities.

The entity also contains entity metadata which describes the entity. Entity metadata is not persisted to the database. It is used by the persistence layer to manage the entity from when it is loaded until it is invoked at runtime. Metadata can be expressed as annotations on the Java programming elements or in XML files (descriptors).

Beginning with the current release, you can define and use extensible entities where mappings can be added on the fly. In this case, the entity stores extended attributes within a map instead of static attributes. The entity then defines how values from this map are mapped to the database using an `eclipselink-orm.xml` mapping file. In addition to being able to dynamically define mappings, TopLink also allows these extended mappings to be stored and managed externally. This external storage allows your extended mappings to be defined while the application is running. For more information, see "EclipseLink/Examples/JPA/Extensibility" in the EclipseLink documentation:

`http://wiki.eclipse.org/EclipseLink/Examples/JPA/Extensibility`

2.1.1.2 Persistence and Persistence Units

Persistence is a characteristic of an entity. This means that the entity can be represented in a data store, and it can be accessed at a later time.

A persistence unit identifies a persistable unit and defines the properties associated with it. It also defines the objects that need to be persisted. The objects can be entity classes, embeddable classes, or mapped superclasses. The persistence unit provides the configuration for the entity manager factory. Entity managers created by the entity manager factory will inherit the properties defined in the persistence unit.

2.1.1.3 Entity Managers

An entity manager enables API calls to perform operations on an entity. Until an entity manager is used to create, read, or write an entity, the entity is just a nonpersistent Java object. When an entity manager obtains a reference to an entity, that entity becomes managed by the entity manager. The set of managed entity instances within an entity manager at any given time is called its persistence context; only one Java instance with the same persistent identity may exist in a persistence context at any time.

You can configure an entity manager to read or write to a particular database, to persist or manage certain types of objects, and to be implemented by a specific persistence provider. The persistence provider supplies the implementation for JPA, including the `EntityManager` interface implementation, the Query implementation, and the SQL generation.

Entity managers are provided by an `EntityManagerFactory`. The configuration for an entity manager is bound to the `EntityManagerFactory`, but it is defined separately as a persistence unit. You name persistence units to allow differentiation between `EntityManagerFactory` objects. This way, your application obtains control over which configuration to use for operations on a specific entity. The configuration that describes the persistence unit is defined in a `persistence.xml` file. You name persistence units to be able to request a specific configuration to be bound to an `EntityManagerFactory`.

2.1.2 Adding Metadata Using Annotations

TopLink provides a set of proprietary annotations as an easy way to add metadata to the Java source code. The metadata is compiled into the corresponding Java class files for interpretation at run time by a JPA persistence provider to manage persistent behavior. You can apply annotations at the class, method, and field levels.

TopLink annotations expose some features of TopLink that are currently not available through the use of JPA metadata.

- basic properties—By default, TopLink persistence provider automatically configures a basic mapping for simple types. Use these annotations to fine-tune the immediate state of an entity in its fields or properties.
- relationships—TopLink defaults some relationships, such as OneToOne and OneToMany. Other relationships must be mapped explicitly. Use the annotations to specify the type and characteristics of entity relationships and to fine-tune how your database implements these relationships.
- embedded objects—An embedded object does not have its own persistent identity; it is dependent upon an entity for its identity. By default, TopLink persistence provider assumes that every entity is mapped to its own table. Use the following annotations to override this behavior for entities that are owned by other entities.

2.1.3 Configuration Files

The following sections describe some of the key configuration files in a TopLink Object Relational Mapping project.

2.1.3.1 About the Default Configuration Values

Oracle TopLink is compliant with the JPA 2.0 specification. The configuration files allow you to change the default values of properties that are defined in the specification. The defaults are extensive and specified in Chapter 10 "Metadata Annotations" in the JPA specification.

`http://jcp.org/en/jsr/detail?id=317`

The configuration is done by exception: if a value is *not* specified in one of the configuration files, then a default value is used.

For the TopLink extensions beyond the JPA specification, the defaults are described in the *Oracle Fusion Middleware Java API Reference for EclipseLink*.

2.1.3.2 Configuring Persistence Units Using persistence.xml

Use the JPA persistence file, `persistence.xml`, to configure the persistence unit. A persistence unit defines the details that are required when you acquire an entity manager. You can specify any vendor-specific extensions in the file by using a `<properties>` element.

This file should appear in the `META-INF/` directory of your persistence unit JAR file or in the classpath.

For more information, see "Configuring Persistence Units Using persistence.xml" in the EclipseLink documentation:

`http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Configuration/JPA/persistence.xml`

2.1.3.3 Specifying TopLink Object-Relational Mappings Using eclipselink-orm.xml

The standard JPA `orm.xml` file is used to apply metadata to the persistence unit. It provides support for all of the JPA 2.0 mappings. You can use this file in place of annotations, or to override JPA annotations in the source code. The EclipseLink `eclipselink-orm.xml` file supports the mappings defined by the `orm.xml` file, plus the full set of EclipseLink extensions beyond JPA 2.0.

For more information on the `eclipselink-orm.xml` file, see "Specifying EclipseLink Object-Relational Mappings Using eclipselink-orm.xml" in the EclipseLink documentation:

`http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Configuration/JPA/eclipselink-orm.xml`

See also "EclipseLink/Examples/JPA/EclipseLink-ORM.XML" in the EclipseLink documentation:

`http://wiki.eclipse.org/EclipseLink/Examples/JPA/EclipseLink-ORM.XML`

Note: Using this mapping file will enable many TopLink advanced features, but it may prevent the persistence unit from being portable to other JPA implementations.

2.1.4 Data Sources

An important part of the definition of the persistence unit is the location where the provider will be able to find data to read and write. This is called the *data source*. In

TopLink, the data source is often a database. The database location is specified in the form of a JDBC data source in the JNDI namespace of the server.

Typically, applications that use TopLink are run in the context of a JTA transaction. Specify the name of the data source in the `jta-data-source` element in the `persistence.xml` file. If the application is not run in the context of a transaction, then it is considered to be *resource-local*. In this case, specify the name of the data source in the `non-jta-data-source` element.

TopLink also allows you to specify a non-relational database data source, such as an XML schema.

Applications that use TopLink can be run in standalone, or *Java SE*, mode. In this mode, the application runs outside the server, with a non-JTA compliant data source, and in a non-Oracle stack. In this case, you must provide driver-specific information, such as the JDBC driver class, the URL that the client uses to connect to the database, and the user name and password needed to access the database. For more information and an example of running an application in stand-alone mode, see "EclipseLink/Examples/JPA/OutsideContainer" in the EclipseLink documentation:

`http://wiki.eclipse.org/EclipseLink/Examples/JPA/OutsideContaine r`

See also "EclipseLink/Examples/JPA/Tomcat Web Tutorial" in the EclipseLink documentation:

`http://wiki.eclipse.org/EclipseLink/Examples/JPA/Tomcat_Web_ Tutorial`

2.1.5 TopLink Caches

By default TopLink uses a shared object cache, that caches a subset of all objects read and persisted for the persistence unit. The TopLink shared cache differs from the local `EntityManager` cache. The shared cache exists for the duration of the persistence unit (`EntityManagerFactory`, or server) and is shared by all `EntityManagers` and users of the persistence unit. The local `EntityManager` cache is not shared, and only exists for the duration of the `EntityManager` or transaction.

The benefit of the shared cache, is that once an object has been read, the database does not need to be accessed if the object is read again. Also, if the object is read by using a query, it will not need to be rebuilt, and its relationships will not need to be re-fetched.

The limitation of the shared cache, is that if the database is changed directly through JDBC, or by another application or server, the objects in the shared cache will be stale.

TopLink offers several mechanism to deal with stale data including:

- Refreshing
- Invalidation
- Optimistic locking
- Cache coordination

The shared cache can also be disabled, or can be selectively enabled and disabled by using the `@Cache` or `@Cacheable` annotations.

TopLink also offers several different caching strategies, to configure how many objects are cached, and how much memory is used.

If the application knows the cache is out of date, it can clear, refresh or invalidate it programmatically. Clearing the cache can cause object identity issues if any of the cached object is in use, so invalidating is safer. If you know that none of the cached objects are in use, then you can just clear the cache.

For more information, see "EclipseLink/Examples/JPA/Caching" in the EclipseLink documentation.

http://wiki.eclipse.org/EclipseLink/Examples/JPA/Caching

2.1.5.1 Defining Cache Behavior

TopLink provides a `@Cache` annotation which allows you to define cache properties. The properties include cache type, size, and refresh rules, among others. See "Using the `@Cache` Annotation" in the EclipseLink documentation.

http://wiki.eclipse.org/EclipseLink/Examples/JPA/Caching#Using_the_.40Cache_Annotation

2.1.5.2 Caching in Clustered Environments

Caching in a clustered environment can have issues, as changes made on one server will not be reflected on objects cached in other servers. This is not a problem, for read-only objects, but it is for objects that are frequently updated.

TopLink offers several solutions to this problem.

- The cache can be disabled for the classes that frequently change.
- Cache coordination can be used to broadcast changes between the servers in the cluster to update of invalidate changed objects.
- Cache invalidation based on time-to-live or time-of-day.
- Optimistic locking will prevent updates to stale objects, and trigger the objects to be invalidated in the cache.

2.1.6 Database Queries

The object-relational component of TopLink supports a variety of queries.

2.1.6.1 JPQL

JPQL is a query language that is similar to SQL, but differs because it presents queries from an object model perspective and includes path expressions that enable navigation over the relationships defined for entities and dependent objects. TopLink enables you to use JPQL with regular Java objects. In TopLink, JPQL enables you to declare queries, using the attributes of each abstract entity in the object model.

The disadvantage of JPQL is that dynamic queries require performing string concatenations to build queries dynamically from Web forms or dynamic content. JPQL is also not checked until runtime, making typos more common. These disadvantages are reduced by using the query criteria API, described in the next section.

See "JPQL" in the EclipseLink documentation.

http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Querying/JPQL

See also "EclipseLink/Release/2.1.0/JPAQueryEnhancements" in the EclipseLink documentation.

http://wiki.eclipse.org/EclipseLink/Release/2.1.0/JPAQueryEnhancements

2.1.6.2 Criteria Queries

JPA 2.0 defines a query criteria API to simplify dynamic query creation. Criteria queries can use parameters, and query hints the same as named queries. The query criteria API allows you to write any JQPL query—all JPQL keywords are defined in this API. The criteria API uses a set of Java interfaces to allow queries to be dynamically constructed. It also provides compile time checking for correctness to reduce the number of runtime typos.

See "Criteria Query" in the EclipseLink documentation.

`http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Querying/Criteria`

2.1.6.3 Query Hints

A TopLink query hint allows a JPA Query to be customized or optimized beyond what is available in the JPA specification. Use TopLink JPA query hints to:

- construct a JPA query
- specify a JPA query using the `@QueryHint` annotation

See "Query Hints" in the EclipseLink documentation.

`http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Query_Hints`

2.1.6.4 Advanced TopLink Native Query Support

TopLink provides an expression framework (also known as *TopLink Native Query Support*) with which you can express queries in a database-neutral fashion as an alternative to raw SQL when writing queries not supported by JPQL. TopLink 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.
- 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, TopLink automatically generates the appropriate join statements to return information from both tables.
- Expressions simplify complex operations.

TopLink automatically generates the appropriate SQL from the specified expression.

The expression framework allows you to work with expressions, database queries, call objects, and native queries. For more information on the queries described in the following list, see "Native SQL Queries" in the EclipseLink documentation.

`http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Querying/Native`

- JPA Query Using an EclipseLink `DatabaseQuery`

  A TopLink `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.

- JPA Query Using a TopLink Call Object

  Using the `DatabaseQuery` method `setCall`, you can define your own TopLink `Call` to accommodate a variety of data source options, such as SQL stored procedures and stored functions, EJB QL queries, and EIS interactions.

- Named Parameters in a Native Query

  Using TopLink, you can specify a named parameter in a native query using the TopLink # convention

- JPQL Positional Parameters in a Native Query

  Using TopLink, you can specify positional parameters in a native query using the Java Persistence query language (JPQL) positional parameter convention `?n` to specify a parameter by number.

- JDBC-Style Positional Parameters in a Native Query

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

2.2 Building Blocks for Object-XML Mapping

The TopLink Object-XML component enables you to efficiently bind Java classes to XML schemas. Object-XML implements JAXB, allowing you to provide your mapping information through annotations as well as providing support for storing the mappings in XML format.

JAXB (Java Architecture for XML Binding—JSR 222) is the standard for XML Binding in Java. JAXB covers 100% of XML Schema concepts. TopLink provides a JAXB implementation with many extensions.

When using TopLink Object-XML as the JAXB provider, no metadata is required to convert your existing object model to XML. You can supply metadata (using annotations or XML) if you want to fine-tune the XML representation.

TopLink Object-XML includes many advanced mappings that allow you to handle complex XML structures without having to mirror the schema in your Java class model.

For more information, see "The EclipseLink MOXy (JAXB) User's Guide" in the EclipseLink documentation:

`http://wiki.eclipse.org/EclipseLink/UserGuide/MOXy`

The following sections describes many of these features.

- Section 2.2.1, "Using TopLink Object-XML as the JAXB Provider"
- Section 2.2.2, "Object-XML Architecture"
- Section 2.2.3, "Serving Metadata for Object-XML"
- Section 2.2.4, "XML Bindings"
- Section 2.2.5, "Specifying TopLink Object-XML Mappings Using eclipselink-oxm.xml"
- Section 2.2.6, "Querying Objects by XPath"

2.2.1 Using TopLink Object-XML as the JAXB Provider

To use TopLink Object-XML as your JAXB provider, you must identify the entry point to the TopLink JAXB runtime. This entry point is the `JAXBContextFactory` class.

Create a text file called `jaxb.properties` and enter the path to the `JAXBContextFactory` class as the value of the `javax.xml.bind.context.factory` context parameter, for example:

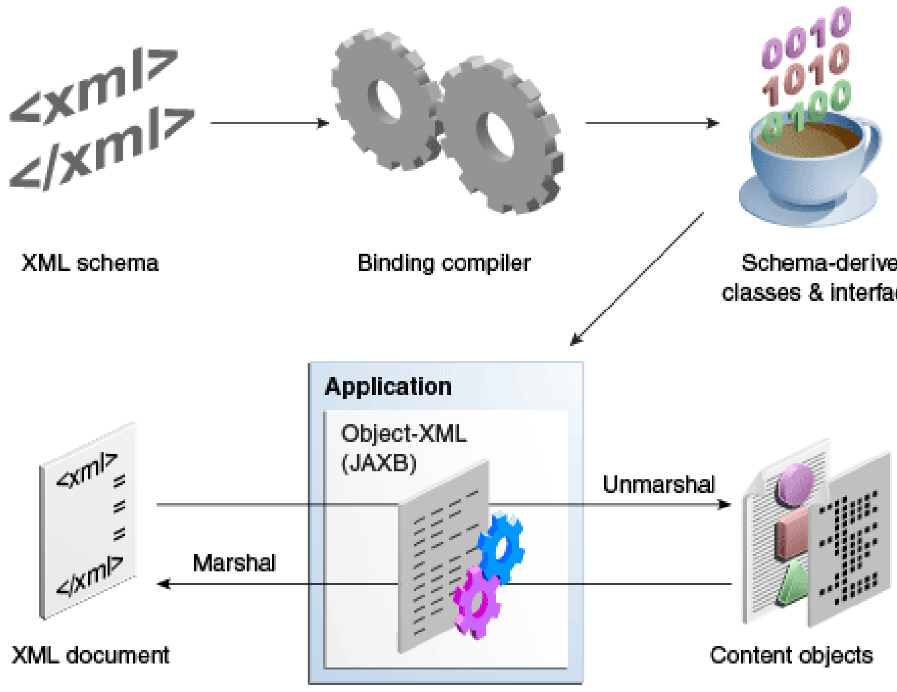
```
javax.xml.bind.context.factory=org.eclipse.persistence.jaxb.JAXBContextFactory
```

The `jaxb.properties` file must appear in the same package as the domain classes.

2.2.2 Object-XML Architecture

In the sample Object-XML architecture illustrated in Figure 2–2, the starting point is a XML schema. A binding compiler binds the source schema to a set of schema-derived program classes and interfaces. JAXB-annotated classes within the application are generated either by a schema compiler or the result of a developer adding JAXB annotations to existing Java classes. The application can either marshal data to an XML document or unmarshal the data to a tree of content objects. Each content object is an instance of either a schema derived or an existing program element mapped by the schema generator and corresponds to an instance in the XML.

Figure 2–2 A Sample Object-XML Architecture



2.2.2.1 JAXB Contexts and JAXB Context Factories

The `JAXBContextFactory` class is the entry point into the TopLink JAXB runtime. It provides the required factory methods and can create new instances of `JAXBContext`.

The `JAXBContextFactory` has the ability to:

- Create a `JAXBContext` from an array of classes and a properties object
- Create a `JAXBContext` from a context path and a classloader

The `JAXBContext` class provides the client's entry point to the JAXB API. The `JAXBContext` class is responsible for interpreting the metadata, generating schema files, and for creating instances of these JAXB objects: `Marshaller`, `Unmarshaller`, `Binder`, `Introspector`, and `Validator`.

TopLink Object-XML offers several options when creating your `JAXBContext`. You have the option of bootstrapping from:

- A list of one or more JAXB-annotated classes.
- A list of one or more TopLink XML Bindings documents defining the mappings for your Java classes.
- A combination of classes and XML Bindings.
- A list of context paths.
- A list of session names, referring to TopLink sessions defined in `sessions.xml`.

2.2.3 Serving Metadata for Object-XML

In addition to the input options described in Section 2.2.2.1, "JAXB Contexts and JAXB Context Factories," TopLink Object-XML provides the concept of a `MetadataSource`, which is responsible for serving TopLink metadata. This allows you to store mapping information outside of your application and have it retrieved when the application's `JAXBContext` is being created or refreshed. For information on implementing `MetadataSource`, see "MetadataSource" in the EclipseLink documentation:

`http://wiki.eclipse.org/EclipseLink/UserGuide/MOXy/Runtime/MetadataSource`

2.2.4 XML Bindings

TopLink allows you to use all of the standard JAXB annotations. In addition to the standard annotations, TopLink offers another way of expressing your metadata—the TopLink XML Bindings document. Not only can XML Bindings separate your mapping information from your actual Java class, it can also be used for more advanced metadata tasks such as:

- Augmenting or overriding existing annotations with additional mapping information.
- Specifying all mappings information externally, without using Java annotations.
- Defining your mappings across multiple Bindings documents.
- Specifying "virtual" mappings that do not correspond to concrete Java fields.

For more information, see "XML Bindings" in the EclipseLink documentation:

`http://wiki.eclipse.org/EclipseLink/UserGuide/MOXy/Runtime/XML_Bindings`

2.2.5 Specifying TopLink Object-XML Mappings Using eclipselink-oxm.xml

You can use Java annotations to specify JAXB features in your TopLink projects. In addition to Java annotations, TopLink provides an XML mapping configuration file called `eclipselink-oxm.xml`. This mapping file contains the standard JAXB mappings, plus configuration options for advanced mapping types. You can use the `eclipselink-oxm.xml` file in place of, or to override JAXB annotations in source code.

Note: Using this mapping file will enable many TopLink advanced features but it may prevent the model from being portable to other JAXB implementations.

2.2.6 Querying Objects by XPath

In addition to using conventional Java access methods to get and set your object's values, TopLink Object-XML also allows you to access values using an XPath statement. There are special APIs on TopLink's `JAXBContext` to allow you to get and set values by XPath. For more information, see "Querying Objects by XPath" in the EclipseLink documentation.

`http://wiki.eclipse.org/EclipseLink/UserGuide/MOXy/Runtime/Querying_Objects_by_XPath`

3 Development Tools for TopLink

This chapter describes the support for TopLink provided by various development tools.

This chapter contains the following sections:

- Section 3.1, "Oracle JDeveloper"
- Section 3.2, "Oracle Enterprise Pack for Eclipse"
- Section 3.3, "Eclipse"
- Section 3.4, "NetBeans"

3.1 Oracle JDeveloper

Oracle JDeveloper is a Java EE development environment with end-to-end support to develop, debug, and deploy e-business applications and Web Services.

For JDeveloper information and downloads, see:

`http://www.oracle.com/us/products/tools/019657.htm`

JDeveloper includes a number of features to aid in the development of applications that use TopLink. These features include wizards to reverse engineer JPA entities from database tables and to generate EJB 3.0 Session Beans with `EntityManager` injection. It also includes methods for querying JPA entities, and test client generation.

The Oracle JDeveloper's TopLink editor allows you to quickly and easily configure and map your Java classes, EJB, and JPA entities to different data sources, including relational databases and XML schemas without using code. The TopLink editor supports multiple standards, including JPA and EJB 3.0.

The DBWSBuilder design-time utility allows you to generate a TopLink DBWS service descriptor and accompanying files. The utility automatically generates these files from database metadata to derive element-tag names and types. The utility also assembles the files into deployable archives.

For more information on TopLink editor and DBWSBuilder, see JDeveloper on-line help.

3.2 Oracle Enterprise Pack for Eclipse

Oracle Enterprise Pack for Eclipse (OEPE) is a set of plug-ins designed to support Java EE development, where Eclipse is your preferred IDE.

For OEPE information and downloads, see:

http://www.oracle.com/technetwork/developer-tools/eclipse/overview/index.html

OEPE helps you complete the following tasks to create a persistence layer that uses TopLink JPA.

- Configure the Persistence Provider for JPA Projects

  OEPE provides an EclipseLink project facet that you can use in your Eclipse JPA project. Selecting the EclipseLink project facet makes the following changes to your project:

  - the project build path is automatically configured to include EclipseLink persistence provider JAR files shipped with Oracle WebLogic Server 11gR1. Note that, even though the library files are not shipped with earlier versions of Oracle WebLogic Server, you can download them using the facet configuration wizard.
  - the database connection properties specific to EclipseLink can be automatically configured in the `persistence.xml` file of your Eclipse JPA project.

- Generate JPA Entities

  OEPE allows you to generate JPA entities using the OEPE JPA Entity Generation Wizard.

- Annotate Java Classes

  Using OEPE, you can annotate existing Java classes (POJOs) with JPA annotations based on a database schema. When you do, Eclipse will add JPA annotations to the appropriate accessors.

- Use a JPA Entity Diagram Editor

  OEPE provides graphical interfaces for viewing entity relationship within a JPA project. Using the Entity Diagram Editor, you can view and modify relationships between entities, get easy access to the entity source code, and create additional object-relational mappings. Note that the editor lets you edit properties of both entities and their fields. You can also edit the `persistence.xml` file that describes the persistence context.

- Use SQL Schema Viewer

  OEPE allows you to examine your database schema using SQL Schema Viewer that displays tables and relationships between them. The viewer displays tables as table nodes. Each node lists all the columns in a table and shows column data types. The node also provides primary and foreign key indicators in a form of icons. Foreign key relationships between tables are represented by links in a form of arrows.

3.3 Eclipse

The Eclipse IDE provides a number of features and utilities to help you create, run, and maintain applications that use JPA. These capabilities are extended if you install OEPE.

For Eclipse IDE information and downloads, see:

http://www.oracle.com/technetwork/developer-tools/eclipse/overview/index.html

The Dali Java Persistence Tools Project provides extensible frameworks and tools for the definition and editing of object-relational mappings for JPA entities. JPA mapping support focuses on minimizing the complexity of mapping by providing entity generation wizards, design-time validation, and a rich UI for entity and persistence unit configuration.

For more Dali information and downloads, see:

`http://www.eclipse.org/webtools/dali`

Other tools and utilities from the Oracle, open source, and third party vendor communities are available from Eclipse Marketplace.

`http://marketplace.eclipse.org/`

3.4 NetBeans

NetBeans IDE bundles GlassFish Server 3.1.1, which includes Oracle TopLink. The IDE provides full support for JPA-based code development. This support includes entity class wizards for constructing entities and editor hints to ensure that entities conform to the JPA specification. NetBeans also provides a persistence unit editor for constructing a `persistence.xml` file.

For NetBeans information and downloads, see:

`http://netbeans.org/index.html`