



## Dali Java Persistence API Tools

Neil Hauge  
neil.hauge@oracle.com  
Project Lead



# About Neil

- Project Lead of Eclipse Dali JPA Tools Project and WTP PMC member
- Development Tools Lead for Oracle TopLink, Oracle's JPA implementation and the basis of the open source *TopLink Essentials* JPA Reference Implementation
- Over six years experience developing Java Object-Relational Mapping tools



# About Java Persistence API (JPA)

- Separate document bundled as part of EJB 3.0 specification
- Suitable for use in different modes
  - Standalone in Java SE environment
  - Hosted within a Java EE Container
- Standardization of current persistence practices
- Merging of expertise from persistence vendors and communities including: TopLink, Hibernate, JDO, EJB vendors and individuals



# JPA—in a Nutshell

- A Java standard that defines:
  - how Java objects are stored in relational databases (specified using a standard set of mappings)
  - a programmer API for reading, writing, and querying persistent Java objects (“Entities”)
  - a full featured query language
  - a container contract that supports plugging any JPA runtime in to any compliant container.



# JPA—POJO Entities

- Concrete classes
- No required interfaces
  - No required business interfaces
  - No required callback interfaces
- `new()` for instance creation
- Direct access or getter/setter methods
  - Can contain logic (e.g. for validation, etc.)
- “Managed” by an EntityManager
- Can leave the Container (become “detached”)



# Object-Relational Mappings

- Core JPA Mappings
  - Id
  - Basic
  - Relationships
    - OneToOne
    - OneToMany/ManyToOne
    - ManyToMany
  - And more...
- Can be specified using Annotations or XML



# JPA Entity—Mappings on Fields

```
@Entity public class Customer {  
  
    @Id  
    private String name;  
    @OneToOne  
    private Account account;  
  
    public String getName() { return name; }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public Account getAccount() { return account; }  
    public void setAccount(Account account) {  
        this.account = account;  
    }  
}
```



# JPA Entity—Mappings on Properties

```
@Entity public class Customer {  
  
    private String name;  
    private Account account;  
  
    @Id  
    public String getName() { return name; }  
    public void setName(String name) {  
        this.name = name;  
    }  
    @OneToOne  
    public Account getAccount() { return account; }  
    public void setAccount(Account account) {  
        this.account = account;  
    }  
}
```

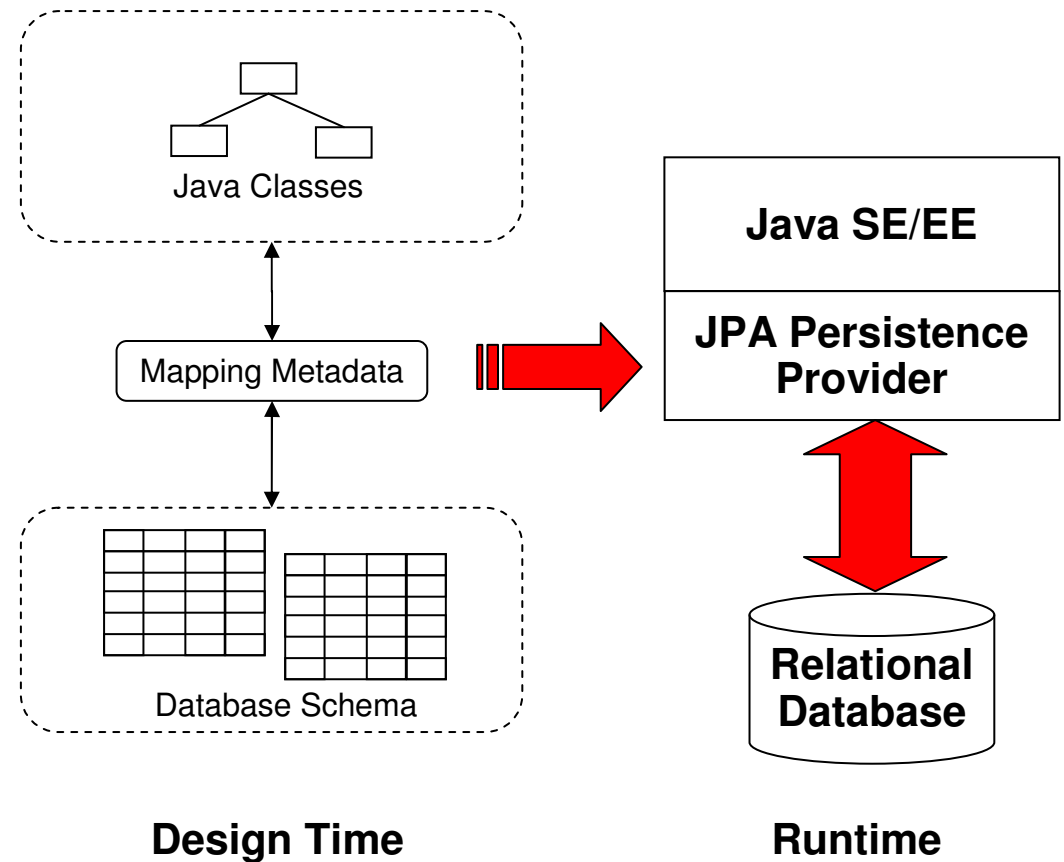


# JPA Entity—Mappings in XML

```
<entity-mappings
  xmlns="http://java.sun.com/xml/ns/persistence/o
  rm"
...
  <entity class="Customer">
    <attributes>
      <id name="name"/>
      <one-to-one name="account"/>
    </attributes>
  </entity>
...
</entity-mappings>
```

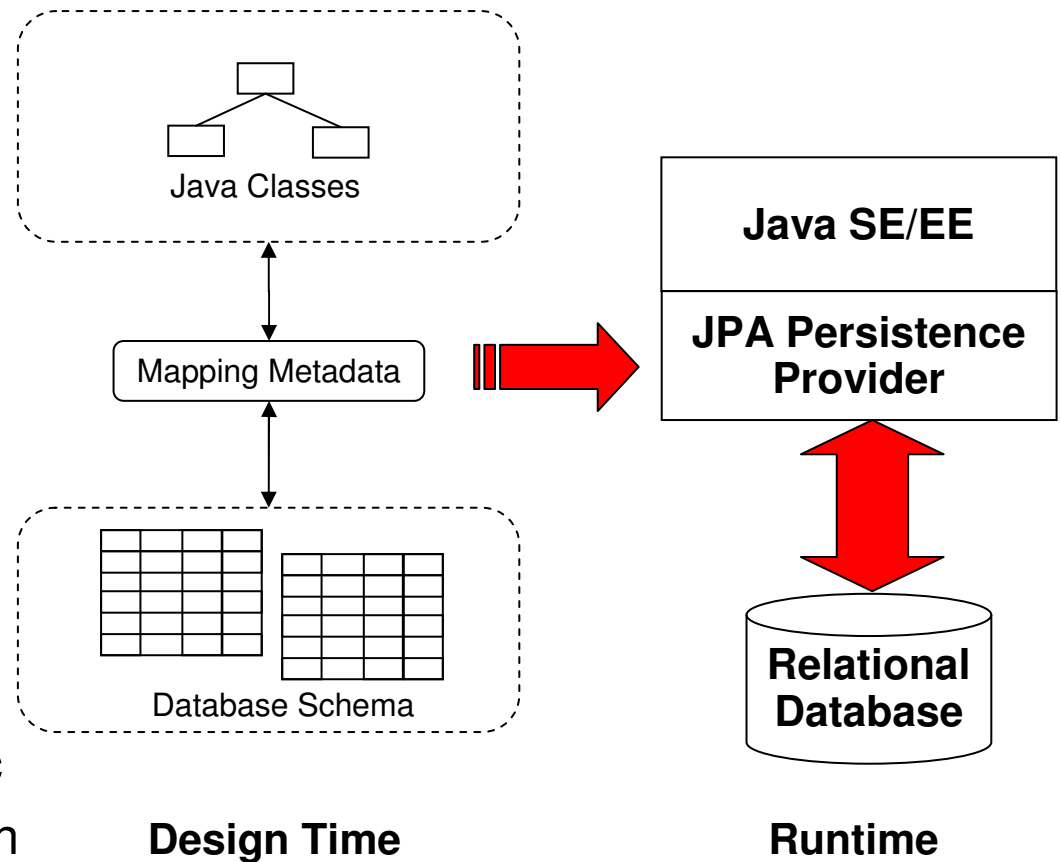
# Why do you need JPA Tools?

- JPA runtime combines:
  - Java Classes
  - Mapping Metadata
  - Database schema



# Why do you need JPA Tools?

- How can you tell if they all match?
  - Deploy and run tests?
    - ✗ slow
    - ✗ find one problem at a time (fix, run, fix, ...)
    - ✓ definitive
  - Design time validation?
    - ✓ quick
    - ✓ finds all issues
    - ✓ validates against spec
    - ✗ runtime may not match spec





# About Dali

- Support for the definition, editing, and deployment of Object-Relational (O/R) mappings for JPA Entities
- Simplify mapping definition and editing through:
  - intelligent mapping assistance
  - dynamic problem identification
  - generation and automated mapping wizards
- Extensible frameworks and tools so vendors and open source projects can provide specific support for their JPA runtimes
- A subproject of the Web Tools Platform (WTP)



# Dali Goals

- **Simplicity**
  - Intelligent mapping assistance and automated generation
- **Intuitiveness**
  - Use existing and consistent modeling and tooling practices in Eclipse
  - Light-weight views offer assistance but don't get in the way of power users
- **Compliance**
  - Support any and all EJB 3.0 compliant runtime implementations
  - Test using EJB 3.0 Reference Implementation
- **Extensibility**
  - Provide the ability for vendors and open source projects to seamlessly add their own value-add features

# Why 'Dali'?

- JPA supports “The Persistence of Memory”—which is the title of a well known Salvador Dali painting.



Salvador Dalí. *The Persistence of Memory*. 1931.  
© 2005 Salvador Dalí, Gala-Salvador Dalí Foundation/Artists Rights Society (ARS), New York



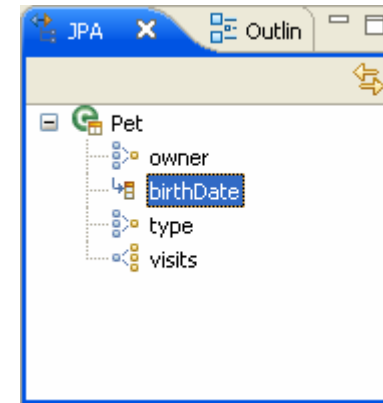
## The Dali Value Proposition

You don't need Dali to build JPA applications—but  
you're going to be way more productive if you do!

ORACLE®

# Dali Contributions to Eclipse

- JPA Mapping Validation
- JPA Structure and Details Views
- Java Source Editor enhancements



```

@Entity
@Table(name="PETS")
public class Pet extends NamedEntity {

    @ManyToOne
    private Owner owner;

    @Basic
    private Date birthDate;
    
```

Problems Error Log

1 error, 0 warnings, 0 infos (Filter matched 1 of 10 items)

Description	Resource
Errors (1 item)	
Column "birthDate" cannot be resolved	Pet.java

JPA Details

Map As: Basic

Column: Default (birthDate)

Table: Default (PETS)

Fetch: Default (Eager)

Optional: Default (False)

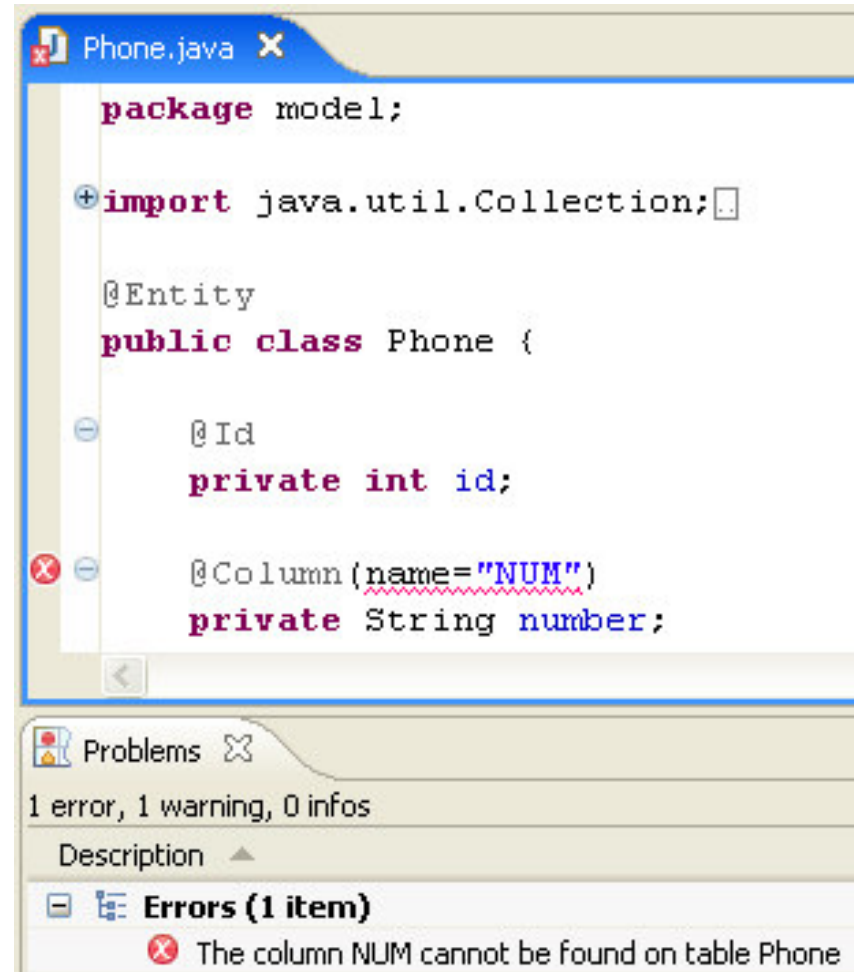
Lob

Temporal:

Enumerated: Default (Ordinal)

# Mapping Validation

- Annotations and/or XML used to define JPA Entities.
- JDT validates syntax, but doesn't understand what the annotations *mean*.



```
Phone.java x
package model;

import java.util.Collection;

@Entity
public class Phone {

    @Id
    private int id;

    @Column(name="NUM")
    private String number;
}
```

Problems X

1 error, 1 warning, 0 infos

Description ▲

Errors (1 item)

- ✖ The column NUM cannot be found on table Phone

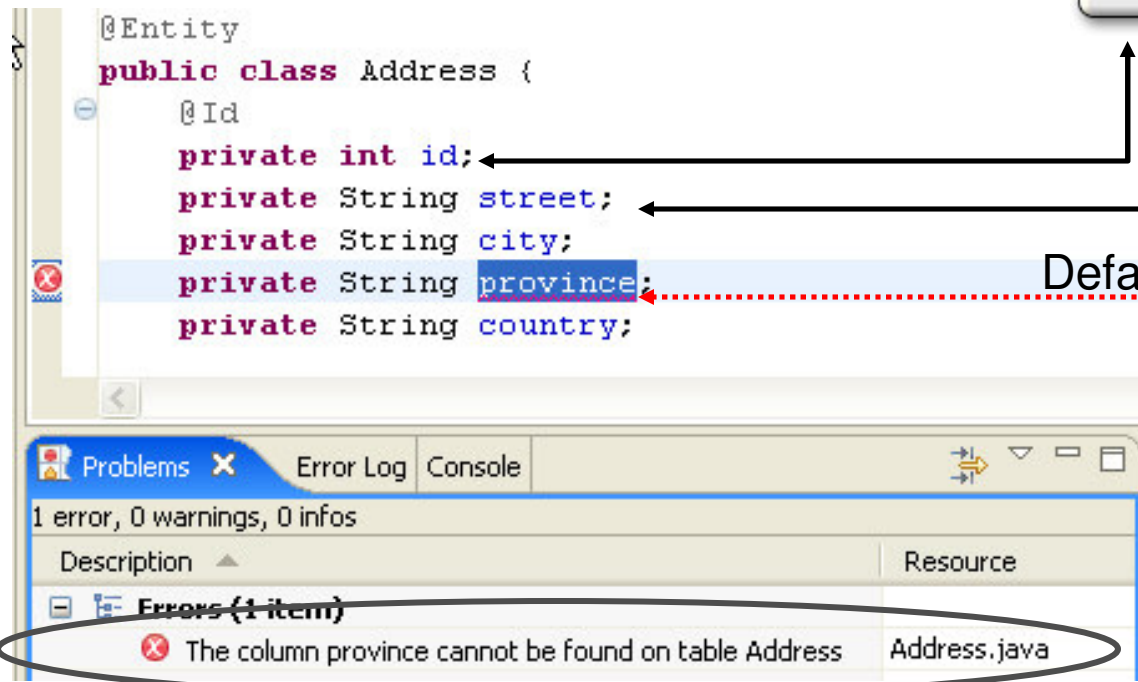
# Mapping Validation

- Java Source Editor enhancements
- Mapping Problem Markers



```

@Entity
public class Address {
    @Id
    private int id;
    private String street;
    private String city;
    private String province;
    private String country;
  }
  
```



The screenshot shows the Java source editor with the `Address` class. A red 'x' icon is next to the `province` field. Below the editor, the 'Problems' window shows one error: 'The column province cannot be found on table Address' in `Address.java`. This error message is circled in red.

Default mapping won't work!

# Mapping Assistance

- JPA Details View



```
@Entity
public class Address {
    @Id
    private int id;
    private String street;
    private String city;
    private String province;
    private String country;
}
```

Persistence Properties

Map As: Default (Basic)

Column:

Name:	Default (province)
Table:	Default (province) CITY COUNTRY
Insertable:	ID
Updatable:	STATE Default (true)

Problems

1 error, 0 warnings, 0 infos

Description	Resource
<b>Errors (1 item)</b>	
The column province cannot be found on table Address	Address.java

# Mapping Assistance for Basic Mapping



```
@Entity
public class Address {
    @Id
    private int id;
    private String street;
    private String city;
    @Column(name="STATE")
    private String province;
}
```

No Mapping Errors!

Persistence Properties

Map As: Default (Basic)

Column:

Name: STATE

Table: Default (Address)

Insertable: Default (True)

Updatable: Default (True)

Problems

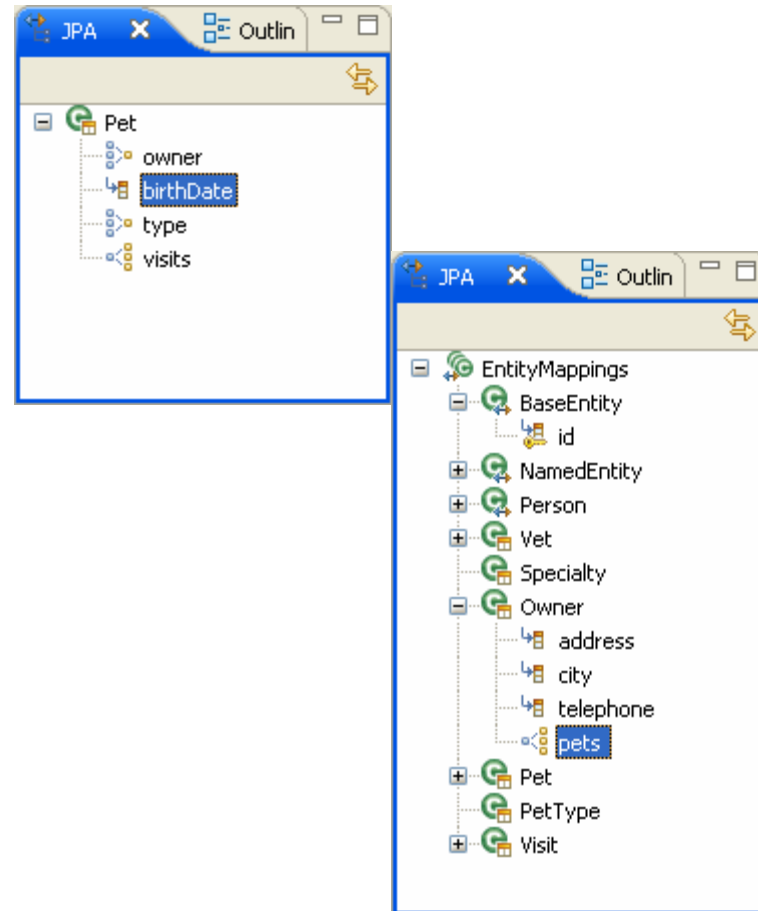
Error Log Console

0 errors, 0 warnings, 0 infos

Description	Resource
-------------	----------

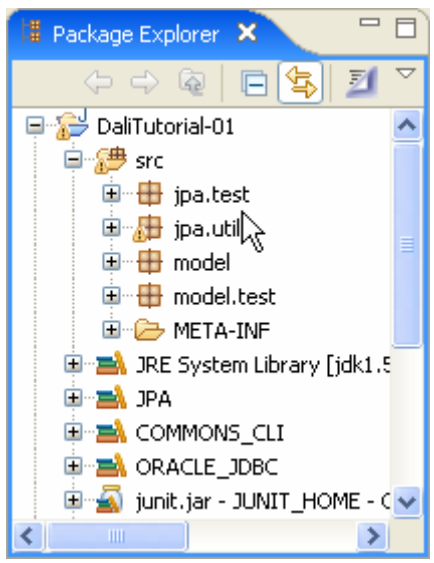
# JPA Structure View

- Provides a JPA specific view of Java Class or ORM XML Mapping File
- A thumbnail sketch of how an Entity is mapped
- Supports navigation between mappings
- Automatically adjusts to either property or field mapping in Java
- Represents structure in Java and XML artifacts





# JPA Perspective—all your JPA Views



```
@Entity
@Table(name="PETS")
public class Pet extends NamedEntity {

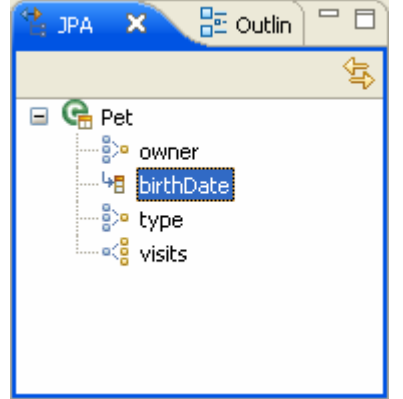
    @ManyToOne
    private Owner owner;

    @Basic
    private Date birthDate;
```

Problems Error Log

1 error, 0 warnings, 0 infos (Filter matched 1 of 10 items)

Description	Resource
Errors (1 item)	
Column "birthDate" cannot be resolved	Pet.java



JPA Details

Map As: Basic

Column: Default (birthDate)

Table: Default (PETS)

Fetch: Default (Eager)

Optional: Default (False)

Lob

Temporal: [dropdown]

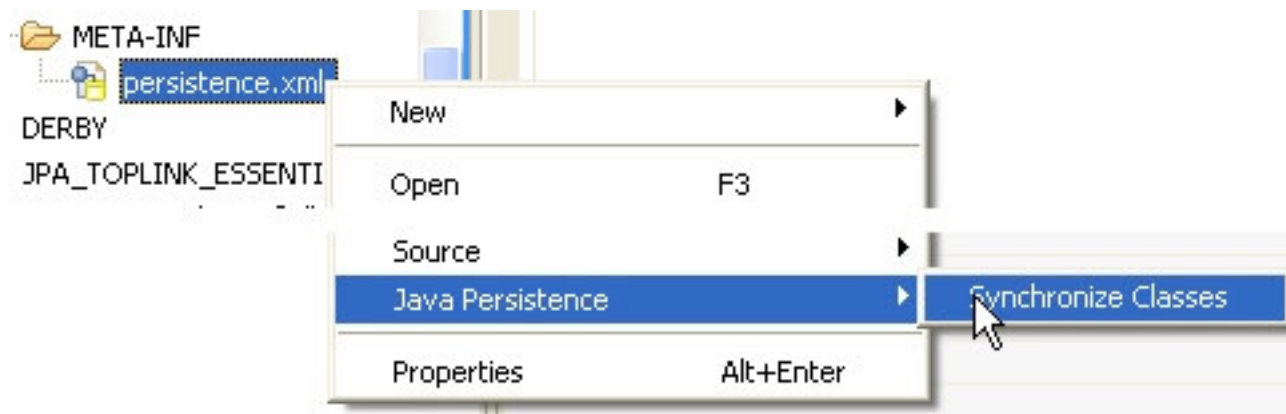
Enumerated: Default (Ordinal)



# Synchronizing Persistence.xml

- In Java SE environment, persistence.xml must list the Entities—Dali offers synchronization

```
persistence.xml x
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  <persistence-unit name="jdb" />
</persistence>
```




# Synchronizing Persistence.xml

```
persistence.xml x
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml
  <persistence-unit name="jdb">
    <class>model.Customer</class>
    <class>model.CustomerInfo</class>
    <class>model.Invoice</class>
    <class>model.Phone</class>
  </persistence-unit>
</persistence>
```

# Integrated F1 Help

- In any Dali view you can hit F1 to get context specific help.



The screenshot shows a help window titled "One-to-one mapping" with the following content:

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

- In the [Persistence Outline view](#), select the field to map. The [Persistence Properties view \(for attributes\)](#) displays the properties for the selected.
- In the Map As field, select **One-to-one**.
- Use this table to complete the remaining fields on the **General** tab in Persistence Properties view.

Property	Description	Default
Target Entity	The entity to which this attribute is mapped.	null You do not need to explicitly specify the target entity, since it can be inferred from the type of object being referenced.
Cascade Type	See " <a href="#">Cascade Type</a> " for details. <ul style="list-style-type: none"> <li>• Default</li> <li>• All</li> <li>• Persist</li> <li>• Merge</li> <li>• Remove</li> </ul>	Default
Fetch Type	Defines how data is loaded from the database.	Eager

Go To: [All Topics](#) [Search](#) [Related Topics](#) [Bookmarks](#) [Index](#)



# Dali 0.5 Release Features

- Support for majority of annotations including relationships
- Integration with WTP RDB components
- JPA Details view supports mapping using database schema with table and column drop downs
- Design time validation of mappings against object and data model (including unspecified default mappings)
- Problems markers (e.g., Entity missing @Id)
- Wizards for basic Entity generation from tables



# Dali 1.0 Features

- Support for XML Mapping File (orm.xml) configuration
  - Uses same views as JPA Annotation editing
- Integration with Data Tools Platform (DTP) for database meta-data
- Facet based functionality for better integration with various WTP project types
- Enhanced validation
- Extensibility for vendor specific extensions

# Demo





# JPA Application Development Scenarios

## Possible Approaches using Dali

- Meet in the Middle
  - existing object and data models
- Bottom Up
  - generate mapped object model from data model
- Top Down
  - generate data model from mapped object model



# Meet In the Middle (MITM)

- The advantage of MITM is that you can focus on getting your object model and data models correct.
  - Table != Class
    - 1 Table could be N classes (using embedded)
    - N tables could be 1 class (with secondary table)
  - Use Java language features like inheritance not present in relational model
- Dali's validation makes MITM practical
  - Avoids map, deploy, debug cycle
  - Provides access to database schema to provide valid choices



# Bottom Up

- Generate Entities from Tables
  - Great way to bootstrap a JPA application from an existing database
  - Uses an Entity == Table approach
  - Do it once and then modify the generated Entities
    - Dali mapping validation will help identify issues resulting from modifications to Entities



# Top Down

- Generate DDL from Entities
  - Some support in Dali 0.5 release
  - Removed in 1.0 in favor of using full featured support implemented by JPA runtimes.
  - Extension point in Dali to allow for plugging in runtime DDL generation.

# Demo





# Dali Adoption

- Oracle
  - Providing majority of resources for Dali
  - Planning to build extensions to Dali for Oracle TopLink and TopLink Essentials runtimes
- RedHat/JBoss
  - Incorporating Dali into JBossIDE as part of Hibernate/JPA toolset
- BEA
  - Considering incorporating Dali into BEA Workshop
- SAP
  - Shipped Dali in Eclipse based SAP NetWeaver Developer Studio Java EE 5 preview
- IBM
  - Incorporating Dali into future tooling projects



# Status of Dali 1.0 (of WTP 2.0)

- High-level goals
  - Support for XML mapping (orm.xml) (Mostly complete)
  - First class component of WTP—integrated as project facet (Complete)
  - Enhanced design-time validation for XML and Java Annotations mappings as well as the combination as defined by the spec. (Complete)
  - Migrate from WTP RDB to Data Tools Project for database access. (Complete)
  - Define API for adopters/extenders to add runtime specific features (e.g., DDL generation) (Still adding)



# Summary

- JPA—the new Java EE standard for object-relational mapping (inside and outside of a container)
- Dali—the WTP project bringing developer productivity to JPA
  - Mapping validation to avoid the map, deploy, debug cycle
  - Intelligent mapping assistance to avoid problems and speed up the process of mapping
  - Integrated with WTP to support development for Java SE and EE

# Getting Started



- Dali JPA Tools—open source tools for JPA development

<http://www.eclipse.org/dali>



- EJB 3.0 & JPA Specification

<http://www.jcp.org/en/jsr/detail?id=220>



- TopLink Essentials—open source JPA Reference Implementation

<http://otn.oracle.com/jpa>  
<http://glassfish.dev.java.net/>



- JPA white papers, tutorials and other resources

<http://otn.oracle.com/jpa>



- Java Developer's Journal—September 2006: Hello Dali!



ORACLE