



ECLIPSEUML STUDIO PRODUCT DOCUMENTATION

ECLIPSEUML

&

ECLIPSE DATABASE

(version 1.1.0)

Contents

EclipseUML

Getting Started	4	State Diagram	101
Class Diagram Creation	4	Concept	101
Classes and Interfaces	9	State Diagram Example	102
Attributes	12	Activity Diagram	103
Methods	15	Concept	103
Associations	20	Drag and Drop	104
Implementation and Inheritance	23	Activity Diagram Example	104
Working With Diagrams	24	Collaboration Diagram	106
Rearranging Diagrams	25	Concept	106
Wire	33	Drag and Drop	106
Zoom	36	Collaboration Diagram example	107
Print	39	Object Diagram	108
Export As Image	40	Concept	108
Customize Perspective	41	Component Diagram	109
Notes	43	Concept	109
Labels	44	Drag and Drop	109
Links	45	Component Diagram Example	110
Key Bindings	46	Deployment Diagram	111
Diagrams	47	Concept	111
Class Diagram	47	Deployment Diagram example	112
Concept	47	Robustness Diagram	113
Drag and Drop	52	Concept	113
Presentation Mode	52	Robustness Diagram example	114
Navigation	52	Preferences	115
Real-Time Synchronization	63	Workbench Preferences	115
Refactoring	64	JDT Preferences	116
Working At Package Level	64	Setting Preferences	118
View Selector	67	Global EclipseUML Preferences	119
Property Concept	70	Class Diagram	127
Toolbar	71	Association	128
Class Diagram Example	72	Class	130
Sequence Diagram	95	Colors	132
Concept	95	Dependency	134
Drag and Drop	96	Inheritance	140
ToolBar	96	Package	140
Reverse Engineering	97	Code generation	143
Sequence Diagram example	98	Documentation generation	144
Use Case Diagram	99	Sequence Diagram	145
Concept	99	J2ee XDoclet Templates	146
UseCase Diagram Example	100	UML Profiles	147
		J2EE Profiles	147
		XDoclet Profiles	181

Reverse Engineering	189		
Getter/setter methods.....	189	Properties View.....	276
Cardinality of association.....	191	Database Connection.....	277
Element type in a collection.....	193	Modeling in the Workspace	289
Inverse association resolution.....	198	Introduction.....	289
Qualified association.....	199	Drag and Drop.....	292
UML model reverse engineering.....	202	Database Connection.....	295
Dependancy detection.....	207	Database Create SQL Script.....	306
Documentation Generation	208	Database Diagram.....	313
Launching the Documentation Generation	208	Database Schema SQL Script.....	318
Templates.....	213	Database Data DTD and XML.....	325
Information.....	213	Forward.....	332
Content.....	213	Index.....	335
Format.....	213	Modeling with the Diagram Editor	341
Getting Started with EMF	216	Introduction.....	341
Class Diagram Creation	216	Concept.....	341
Classes and Interfaces	217	Editor Commands.....	342
Attributes	220	Editor Preferences.....	356
Methods	221	Editor Properties.....	370
Associations	223	Code Generators	409
Implementation and Inheritance.....	226	Introduction.....	409
		Torque Mapping Object Resources.....	409
		Object Relational Bridge Resources.....	419
		Hibernate	425
		Hibernate Configuration and	
		Mapping Resources.....	425
		Hibernate Java Resources.....	434
Overview	227	Extending EclipseDatabase	442
Introduction.....	227	Introduction.....	443
Modeling.....	229	Database Definition.....	443
Code Generation.....	234	Database Definition Update.....	444
Global Preferences	236	New Database Definition.....	445
Introduction.....	236	Hidden Options	446
Database.....	236	Introduction.....	446
Database Connection.....	239	.options.....	447
Database Diagram.....	246	Debug Mode.....	448
Database Schema	248	Database Explorer.....	450
Database Schema DTD.....	251		
Database SQL.....	253		
Templates.....	255		
Database Perspective	259		
Introduction.....	259		
Database Perspective.....	260		
DatabaseExplorer.....	261		
DatabaseConsole.....	265		
Working with DatabaseExplorer	268		
Introduction.....	268		
General Commands.....	269		

EclipseDatabase

Overview	227
Introduction.....	227
Modeling.....	229
Code Generation.....	234
Global Preferences	236
Introduction.....	236
Database.....	236
Database Connection.....	239
Database Diagram.....	246
Database Schema	248
Database Schema DTD.....	251
Database SQL.....	253
Templates.....	255
Database Perspective	259
Introduction.....	259
Database Perspective.....	260
DatabaseExplorer.....	261
DatabaseConsole.....	265
Working with DatabaseExplorer	268
Introduction.....	268
General Commands.....	269

Getting Started

This tutorial is designed to get you started on using the Eclipse UML plugin for Eclipse. The following areas are covered:

[Creating a First Diagram](#)

- [Class Diagram Creation](#)
- [Classes and Interfaces](#)
- [Attributes](#)
- [Methods](#)
- [Associations](#)
- [Implementation and Inheritance](#)

Class Diagram Creation

It is important to always create a src directory.

This section uses the following elements:

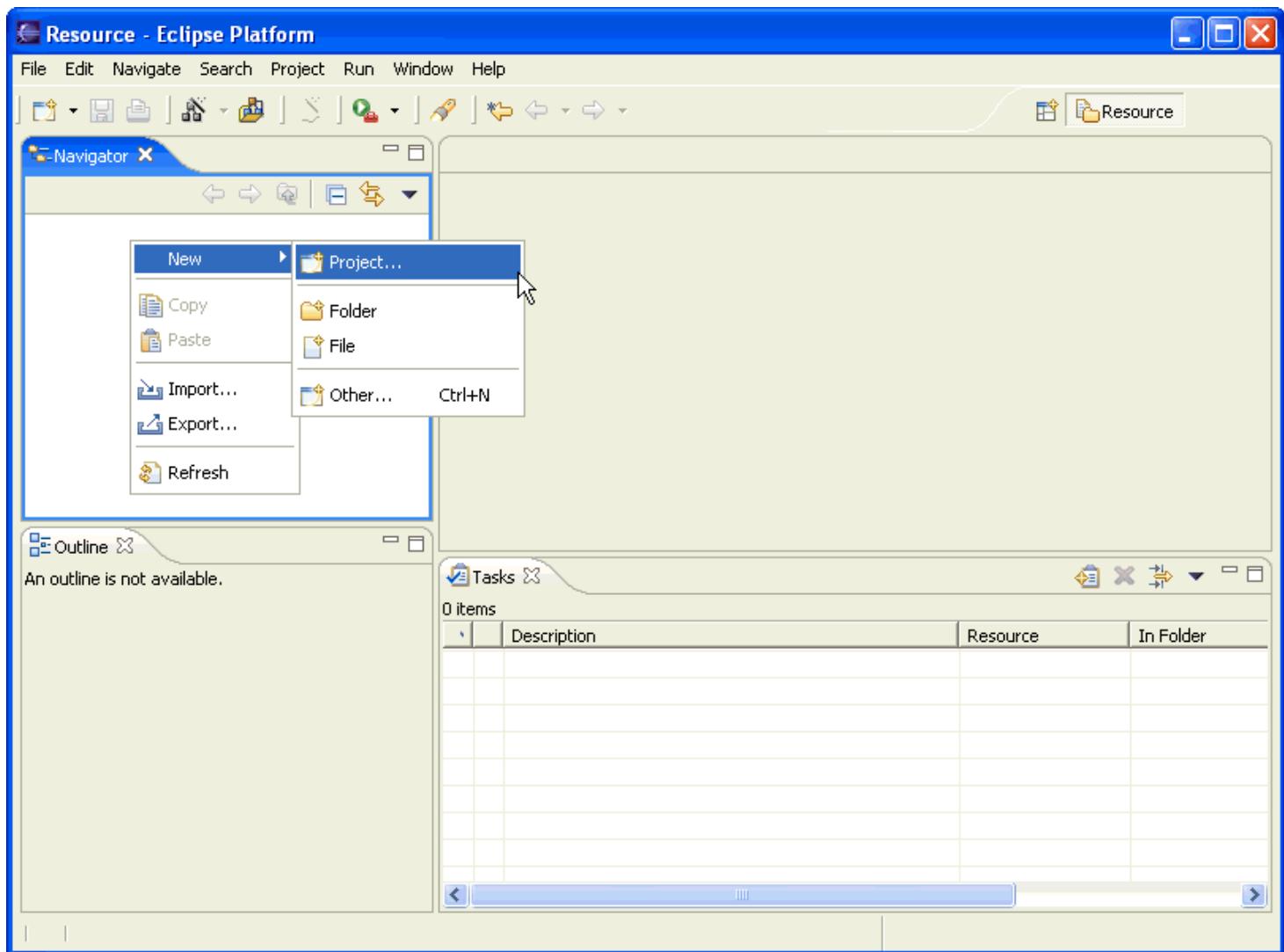
[Create a Java Project](#)

[Create a Package](#)

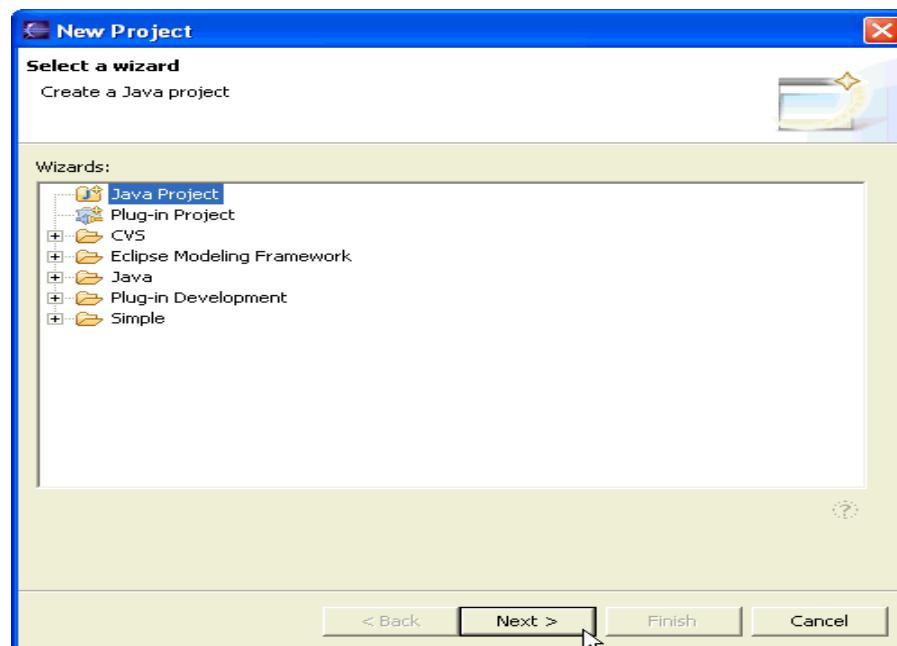
[Create a Class Diagram](#)

1. Create a Java Project

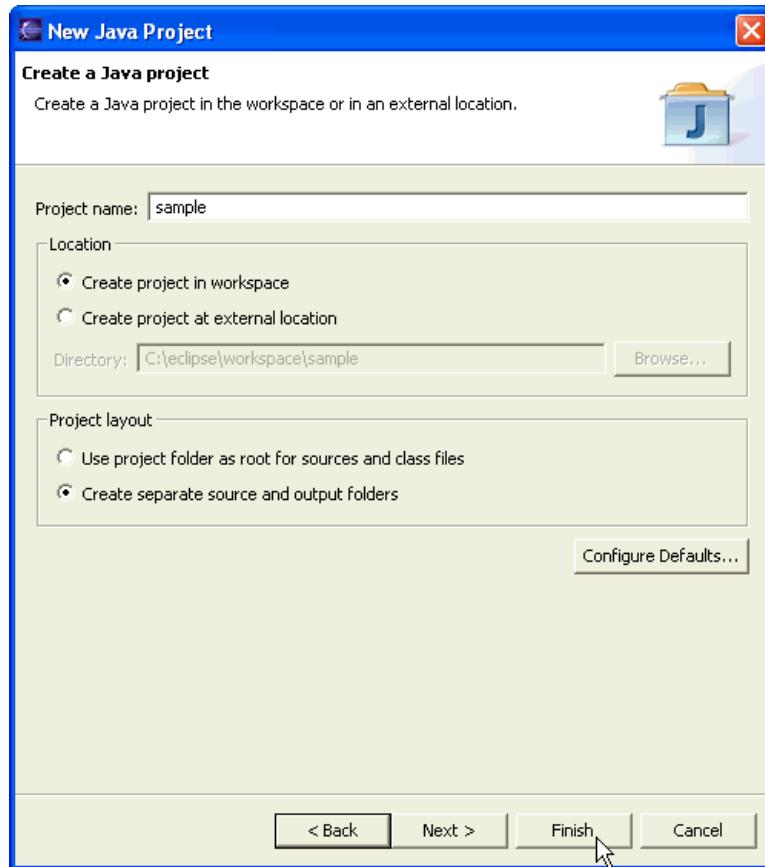
When starting your class diagram creation, you should create a Java Project.
Right click in the navigator window select **New > Project**



Select **Java Project** and click on the Next button

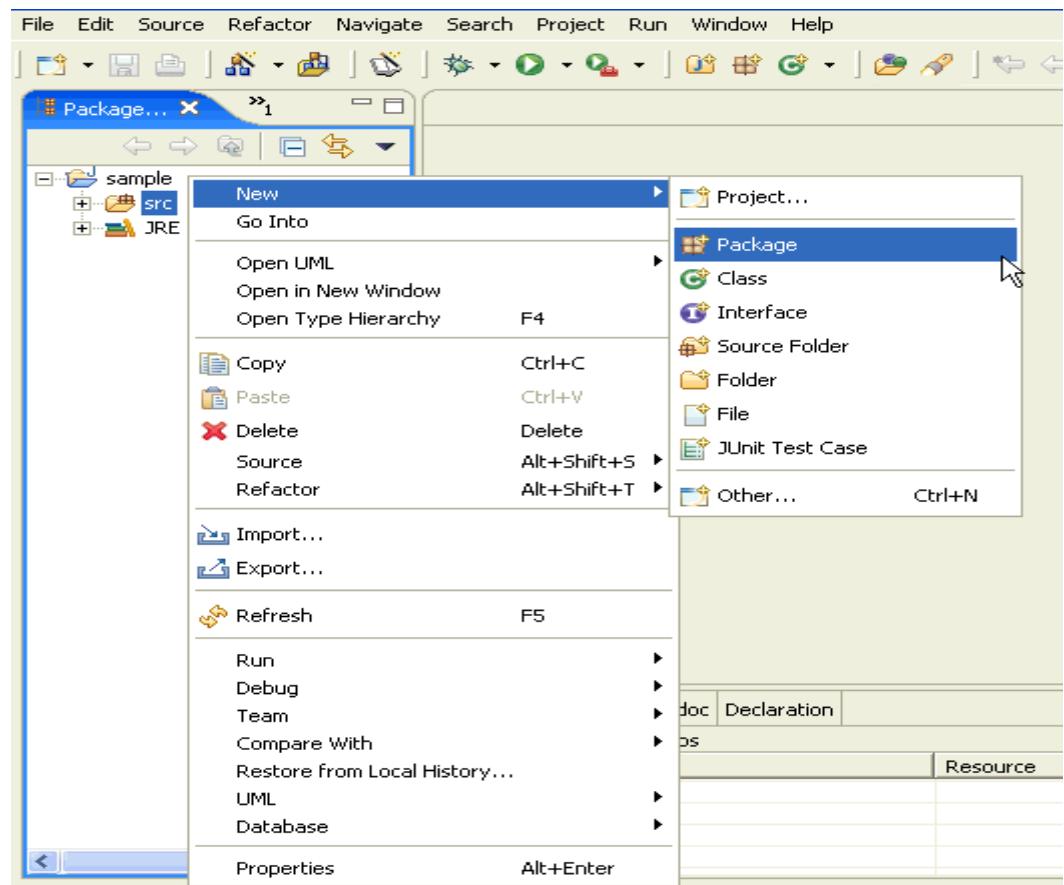


Enter the name of the project in the Project name field.
 Select your workspace location and create separate source and output folders

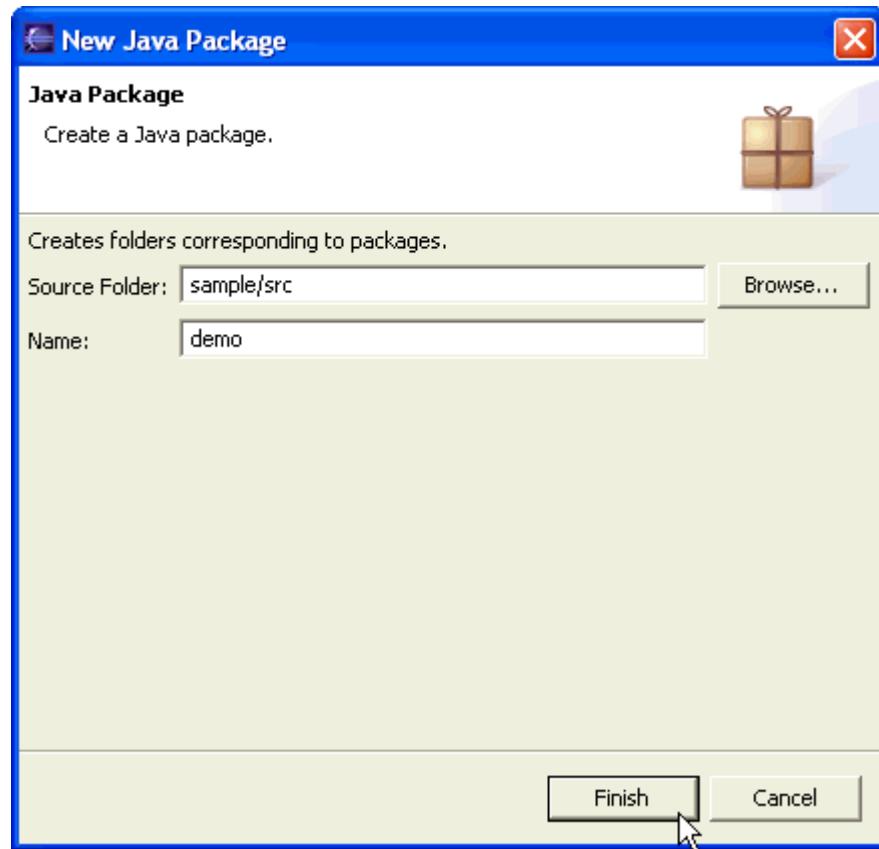


2. Create a new Package

Select src in the Package Explorer and open the folder popup menu New > Package



Enter the name of your package in the Name field and click on the Finish button.

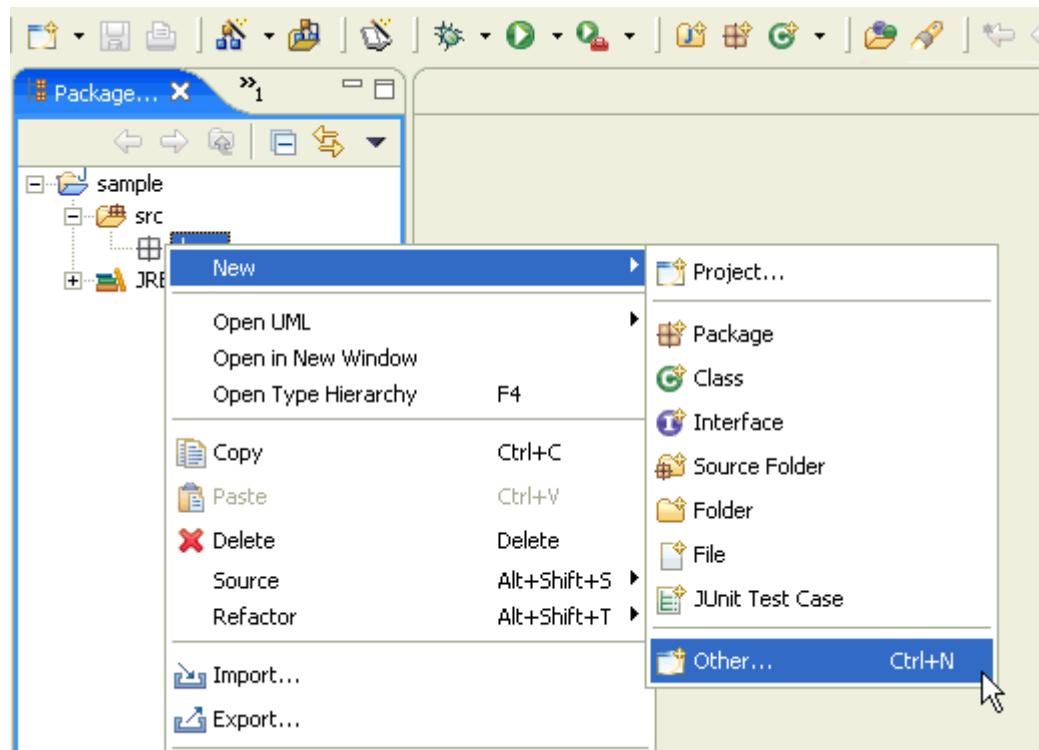


We have successfully created a project, a src folder and a package.



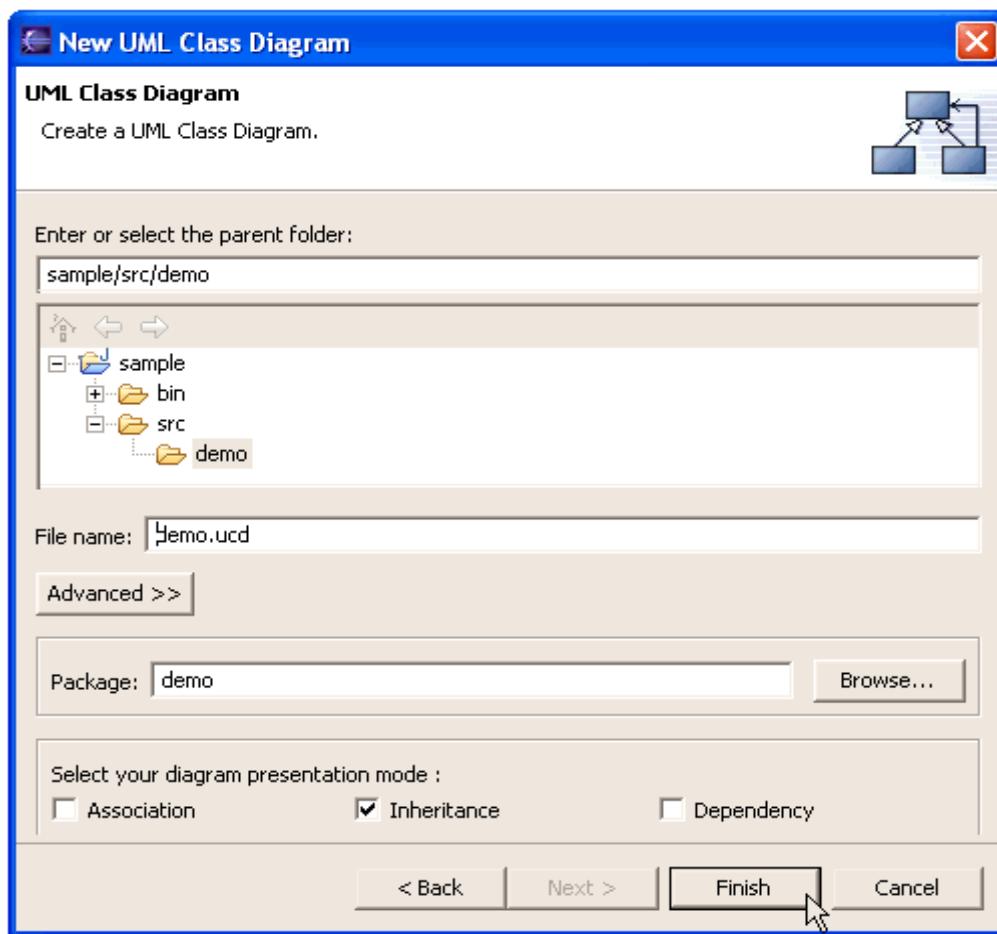
3. Create a Class Diagram

To create a new Class Diagram, select a Package in the Package Explorer, then open the Package popup menu **New > Others**.

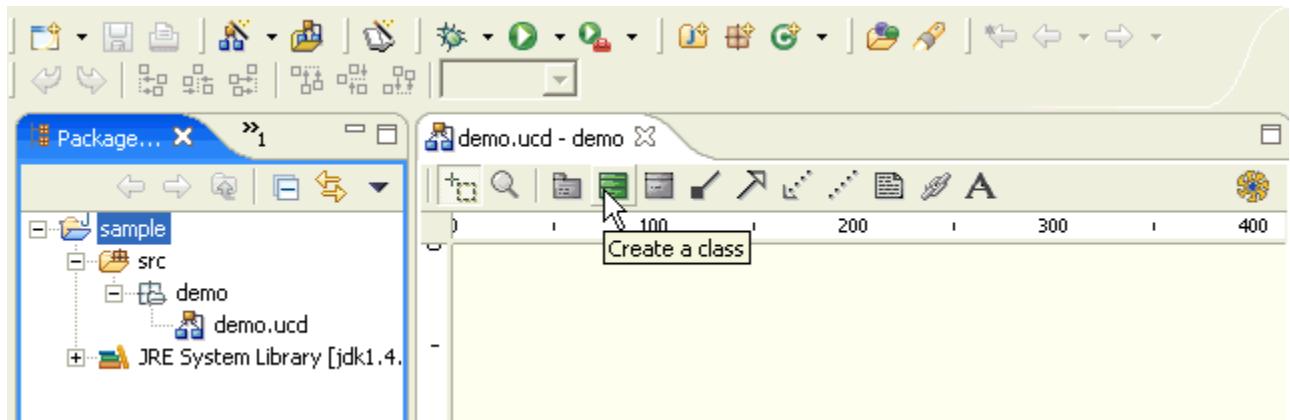


Select **Uml Diagrams > UML Class Diagram**.

Click on the Next button. Select the package you want to work with and enter the name of the Class Diagram in the File name field.

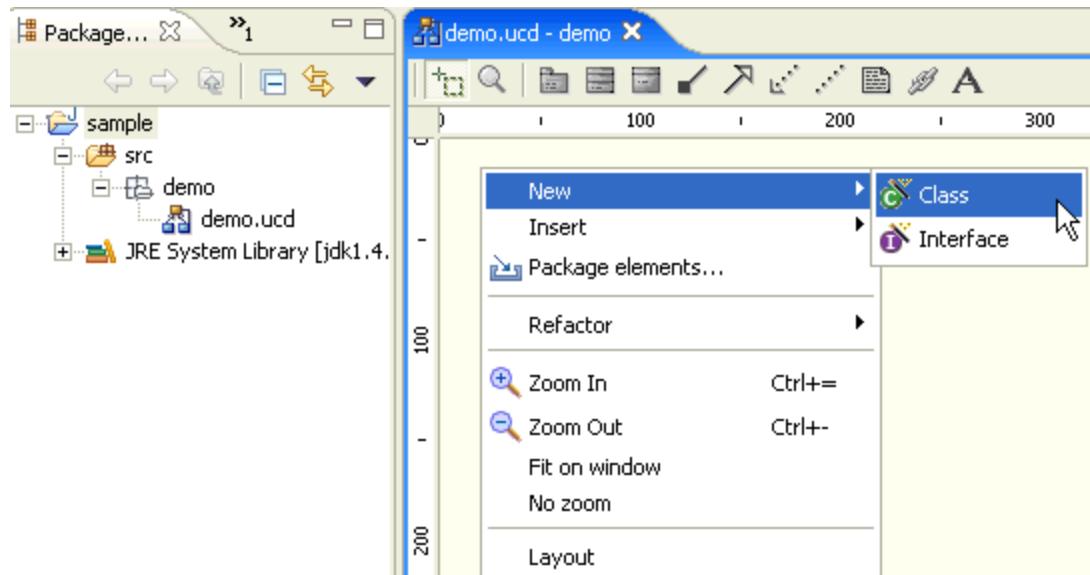


Your new class diagram editor has been created.

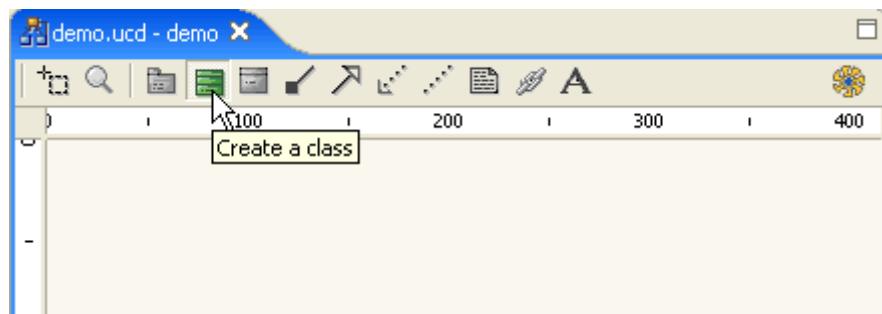


Classes and Interfaces

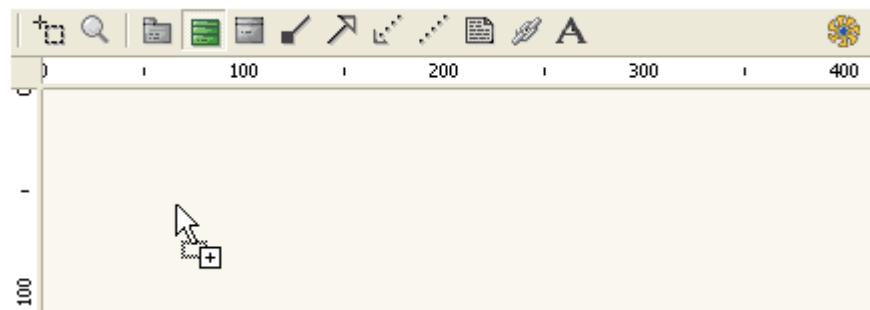
To create a new class (or interface) directly in your class diagram, select *New > Class* (or *New > Interface*) in the class diagram contextual menu.



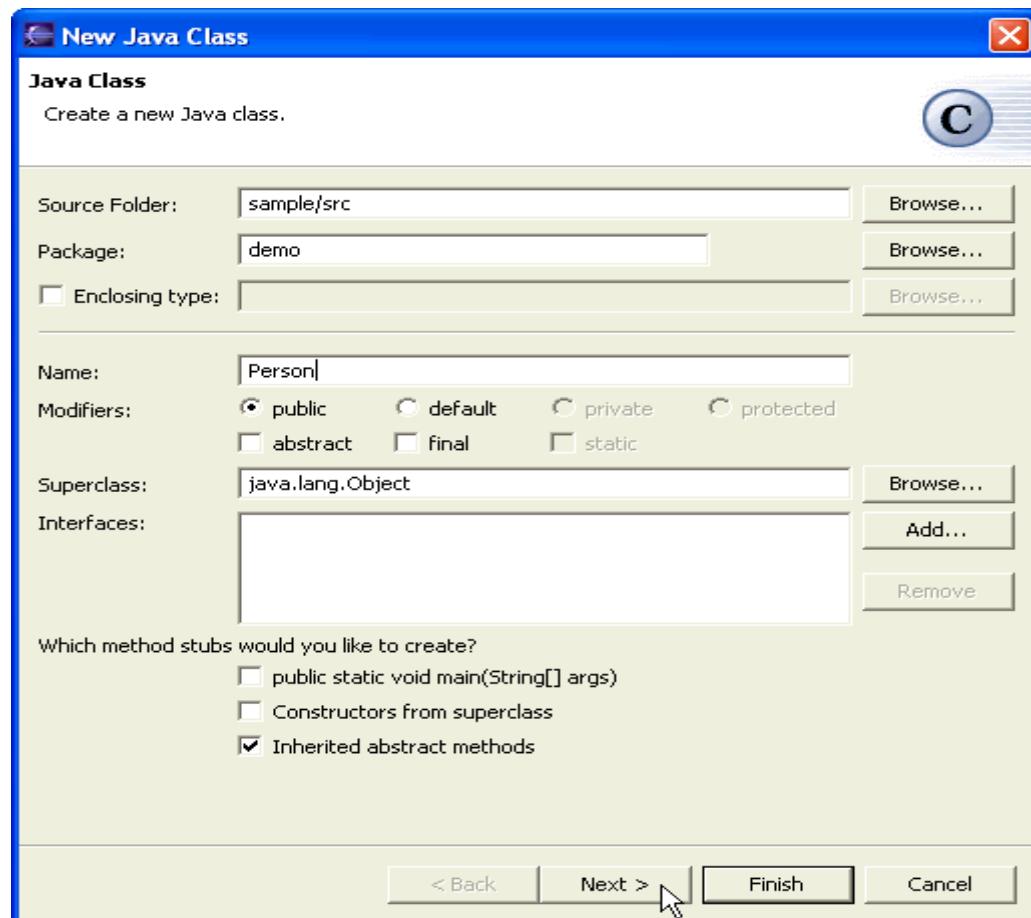
To create a new class is also possible by selecting the green icon in the tool bar (create a class diagram).



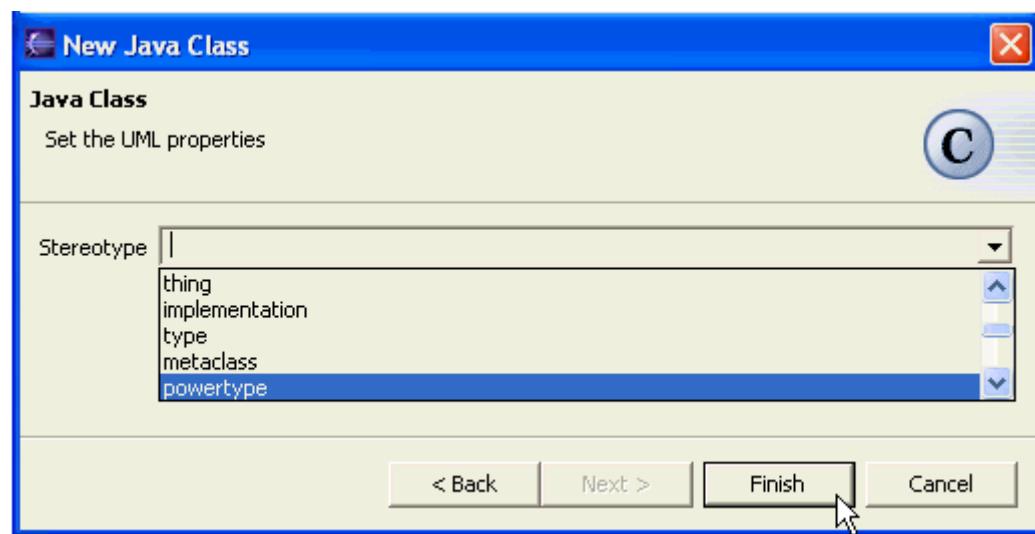
Drag and drop the icon inside your class diagram editor to create a new class.



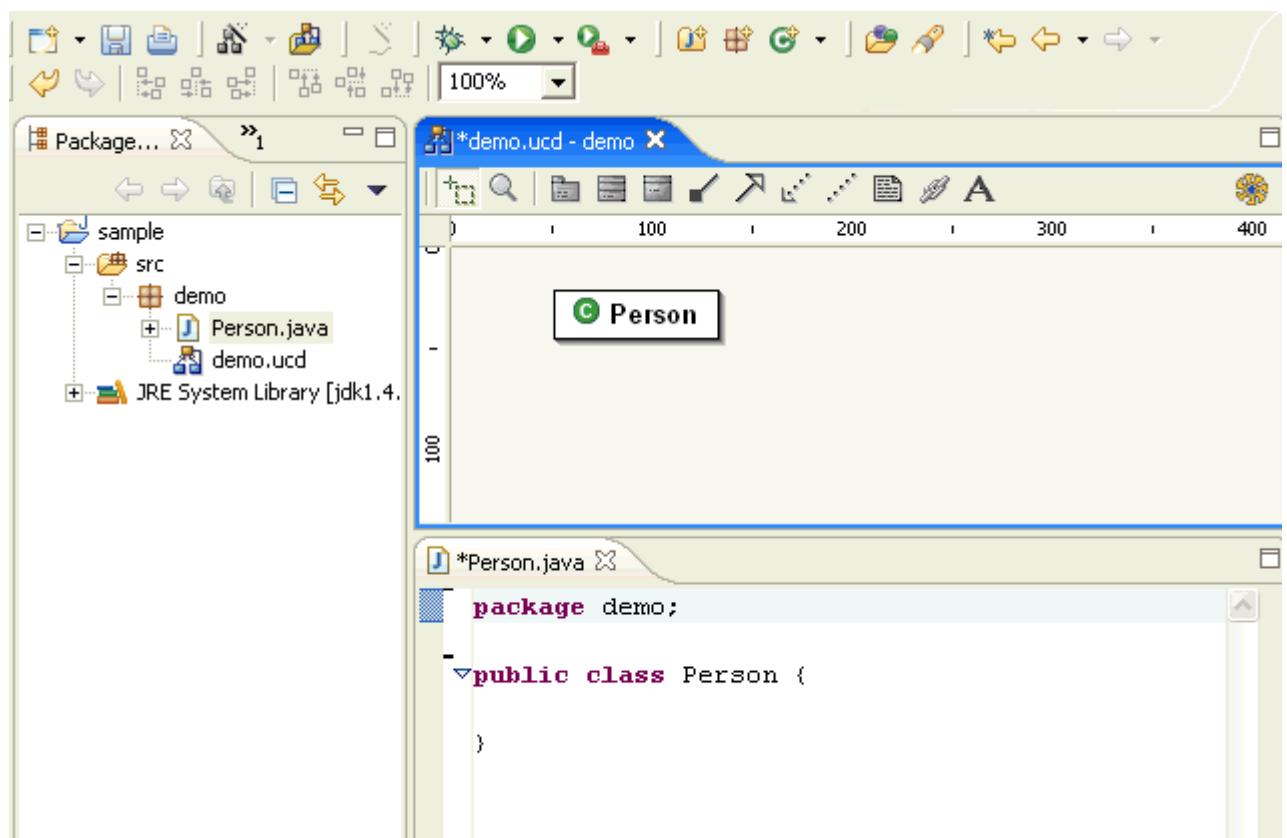
The eclipse class creation dialog (or interface creation dialog) appears.
Enter the name of the class in the Name field and click on the Next button.



Select a stereotype if needed and click on the Finish button.

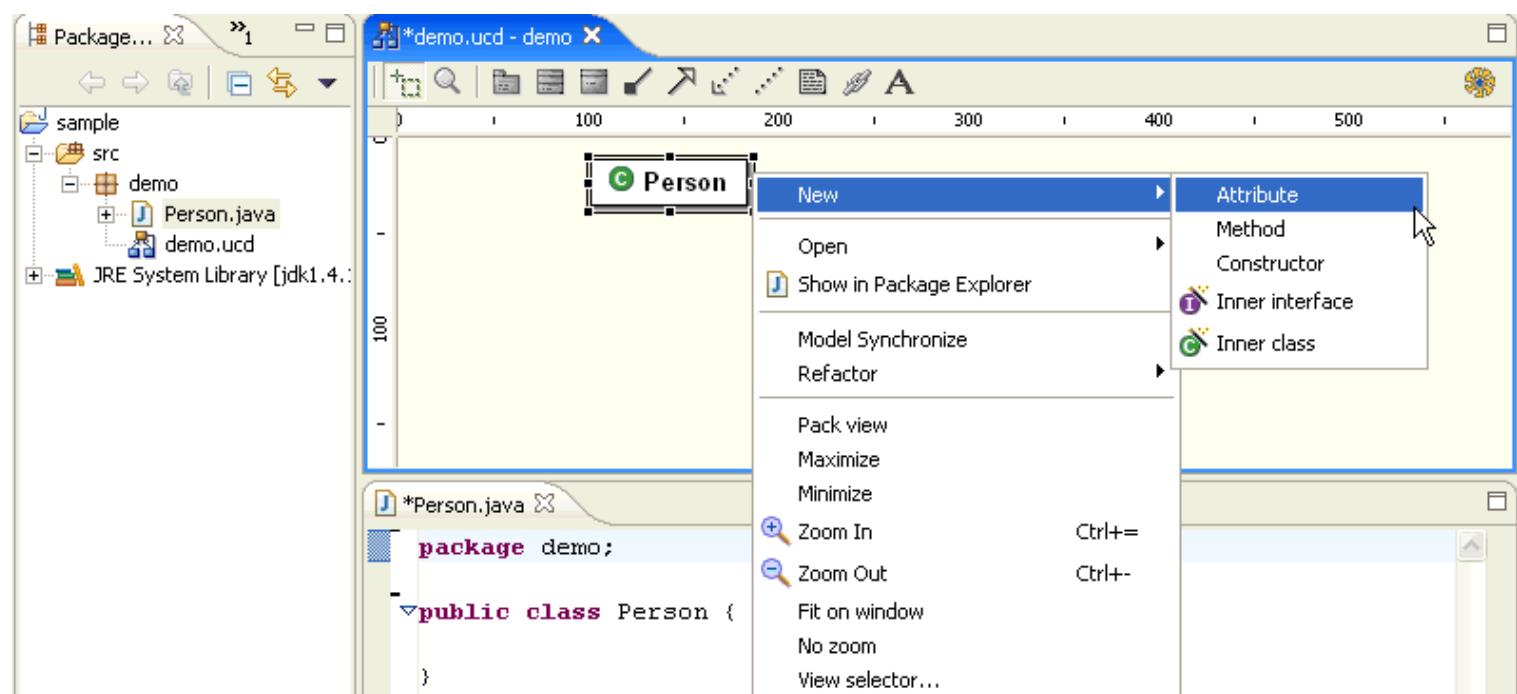


After validating the dialog box, the class appears immediately in the class diagram. If you click on the class, its code appears in the java editor area.

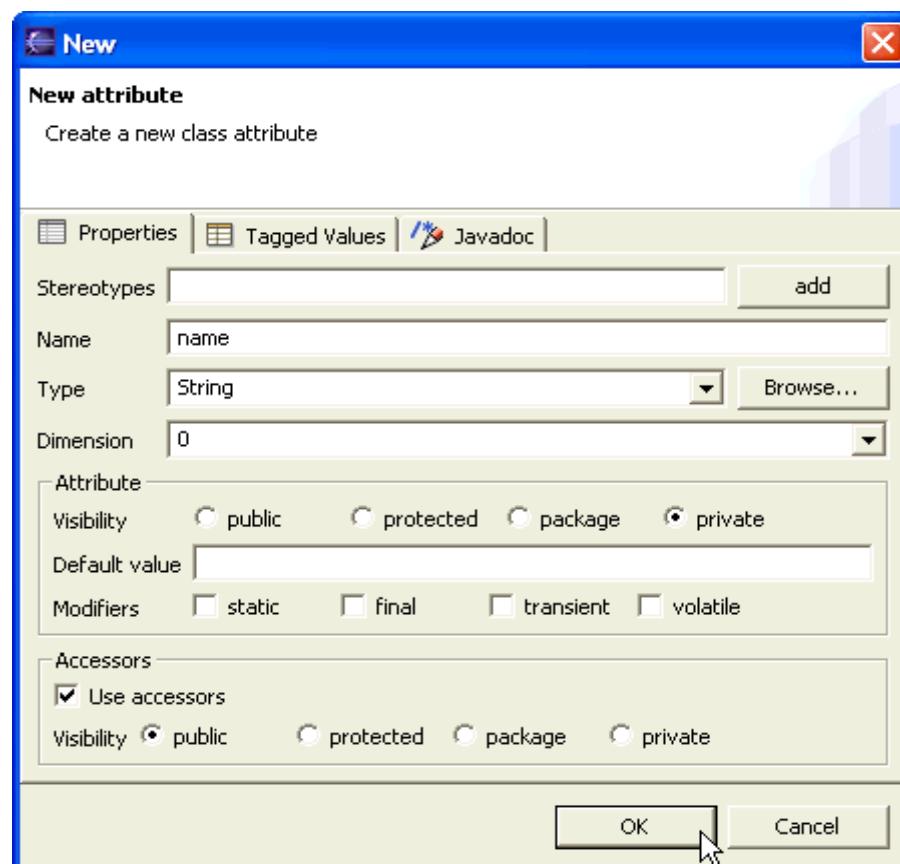


Attributes

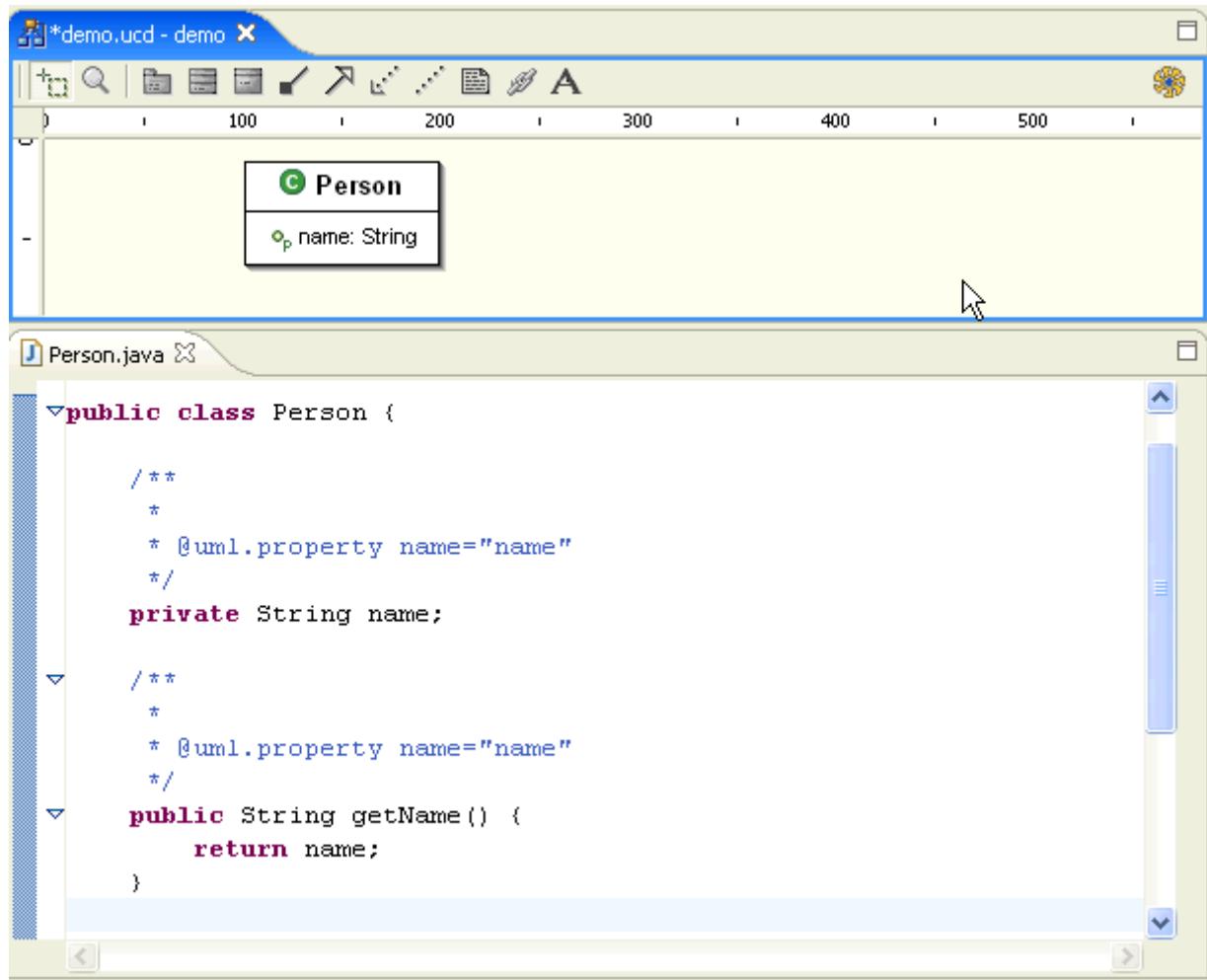
To create a new attribute, select **New > Attribute** in the class (or interface) contextual menu.



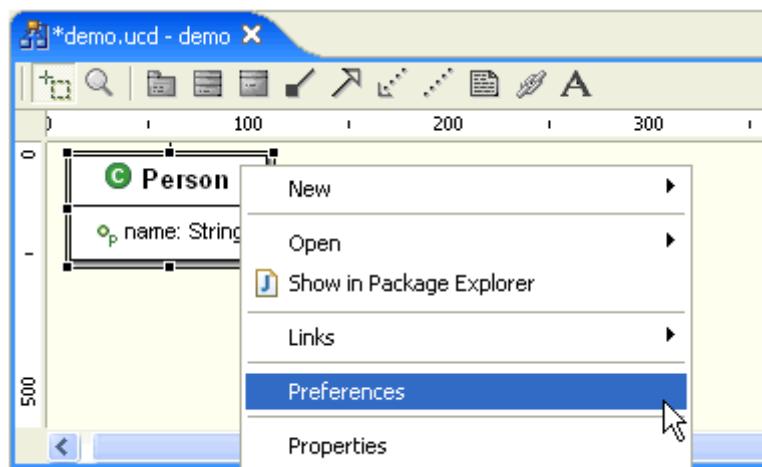
This launches an attribute creation dialog box. You have to set the attribute name, type, multiplicity, default value, visibility and modifiers. For an interface, the attribute is always **static final**.



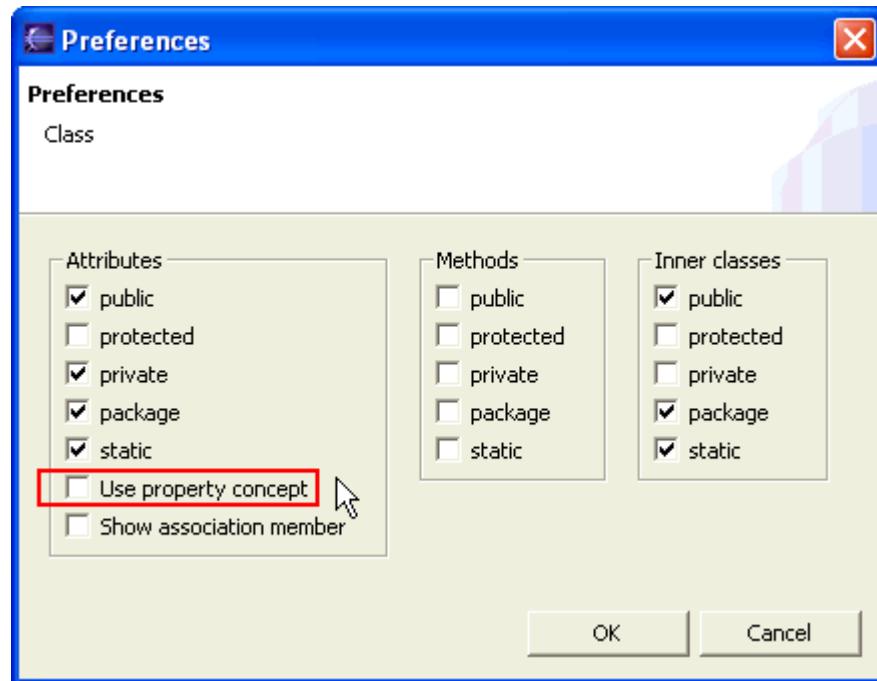
After validating the dialog box, the attribute appears immediately in the class. If you click on the attribute, its code appears in the java editor area. By default, the property concept is activated, so the attribute and its accessors are displayed as a single property. In this example, the attribute is private and is related to its public accessors in the java code.



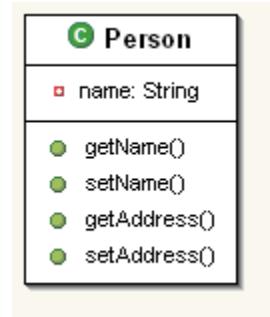
If we don't want to use the [Property concept](#), then select a class in the class diagram editor. Open the popup >Preferences



Unselect the Use property concept checkbox.

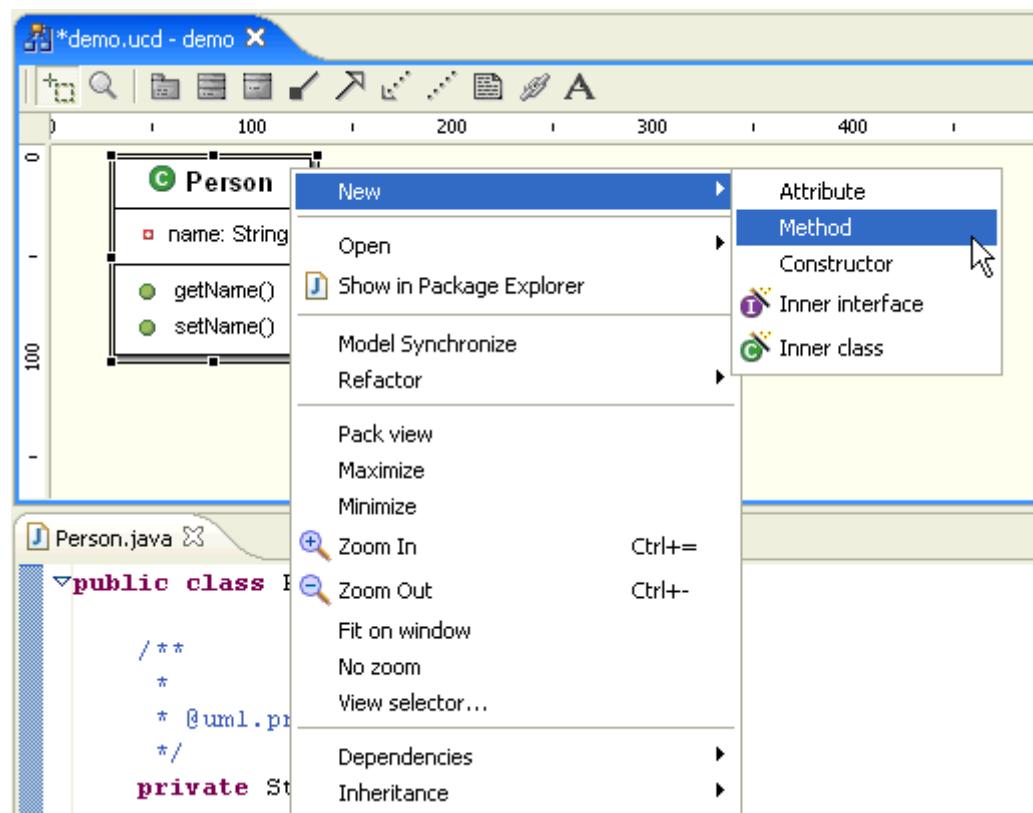


The class diagram editor now shows private attributes and public methods.

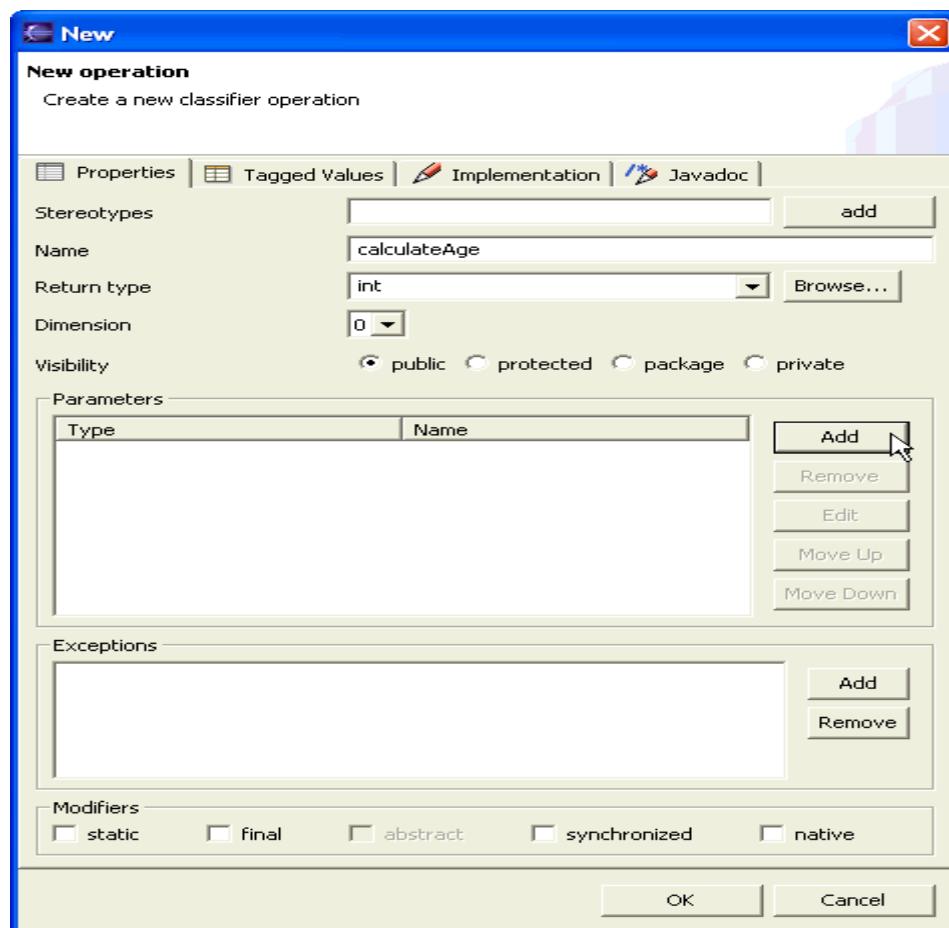


Methods

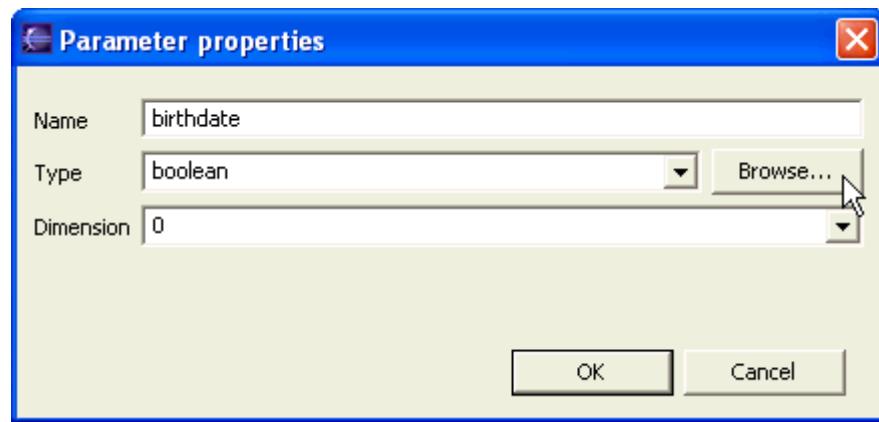
To create a new method, select **New > Method** in the class (or interface) contextual menu.



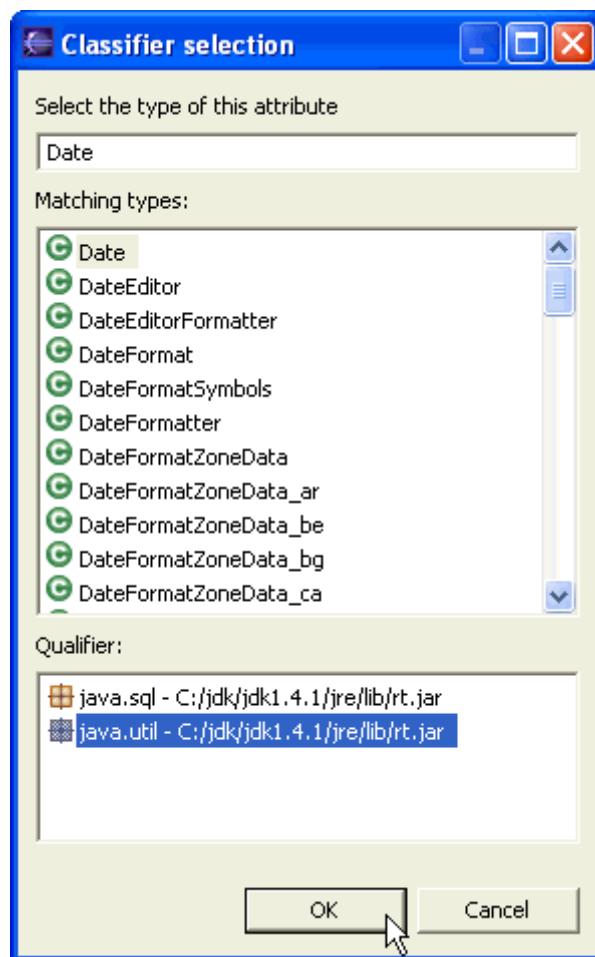
This launches a method creation dialog box. You have to set the method name, return type (with its multiplicity), visibility, parameters, exceptions and modifiers.



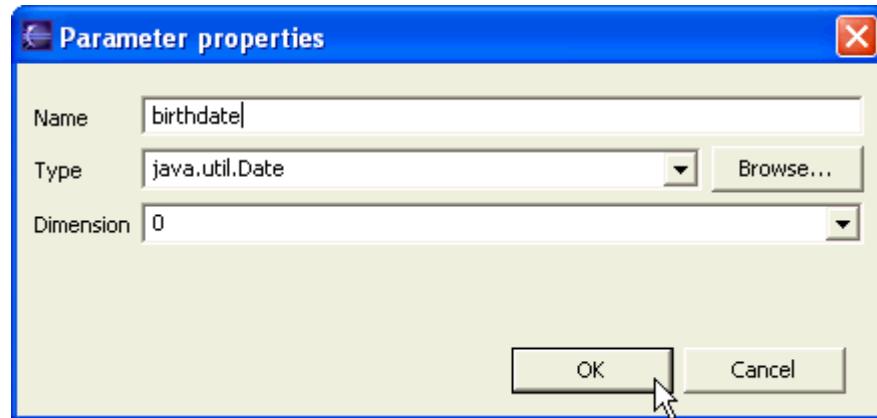
For each parameter, you must indicate its name and type.



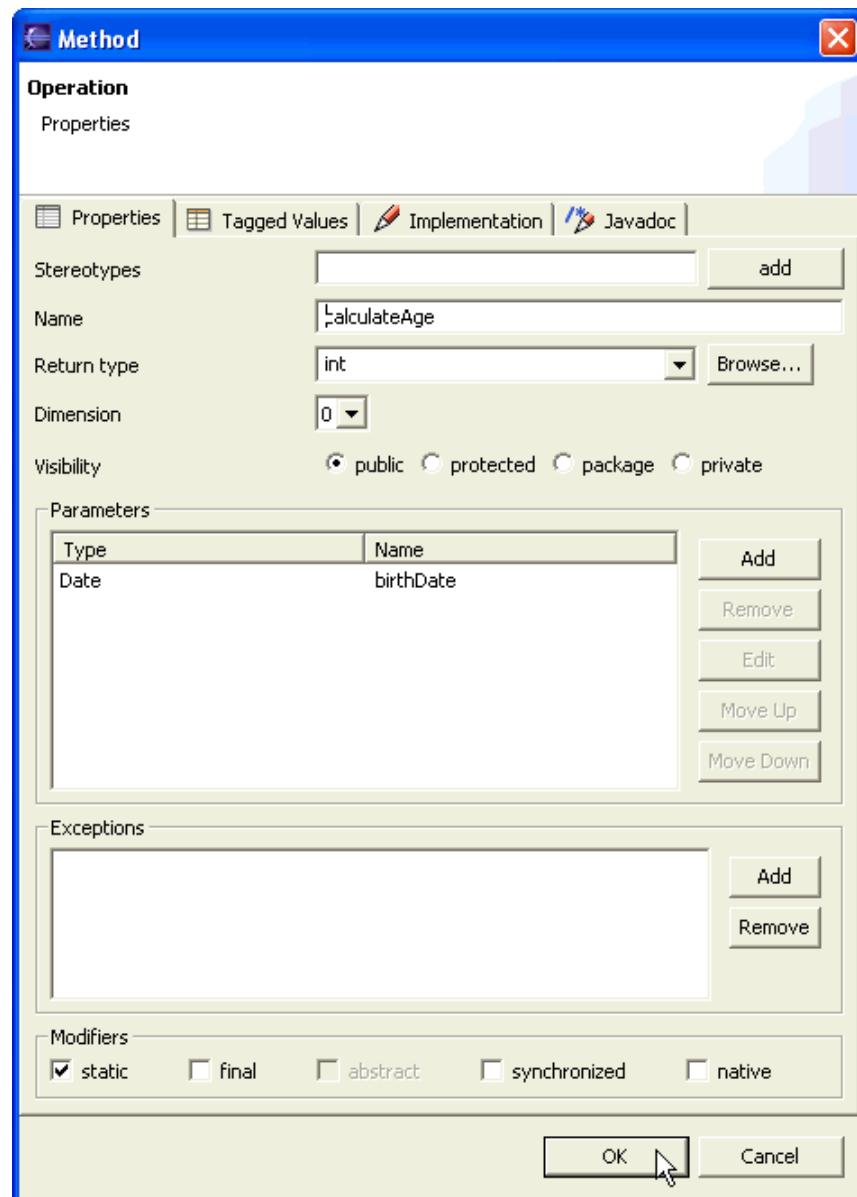
The parameter type is selected through a class finder.



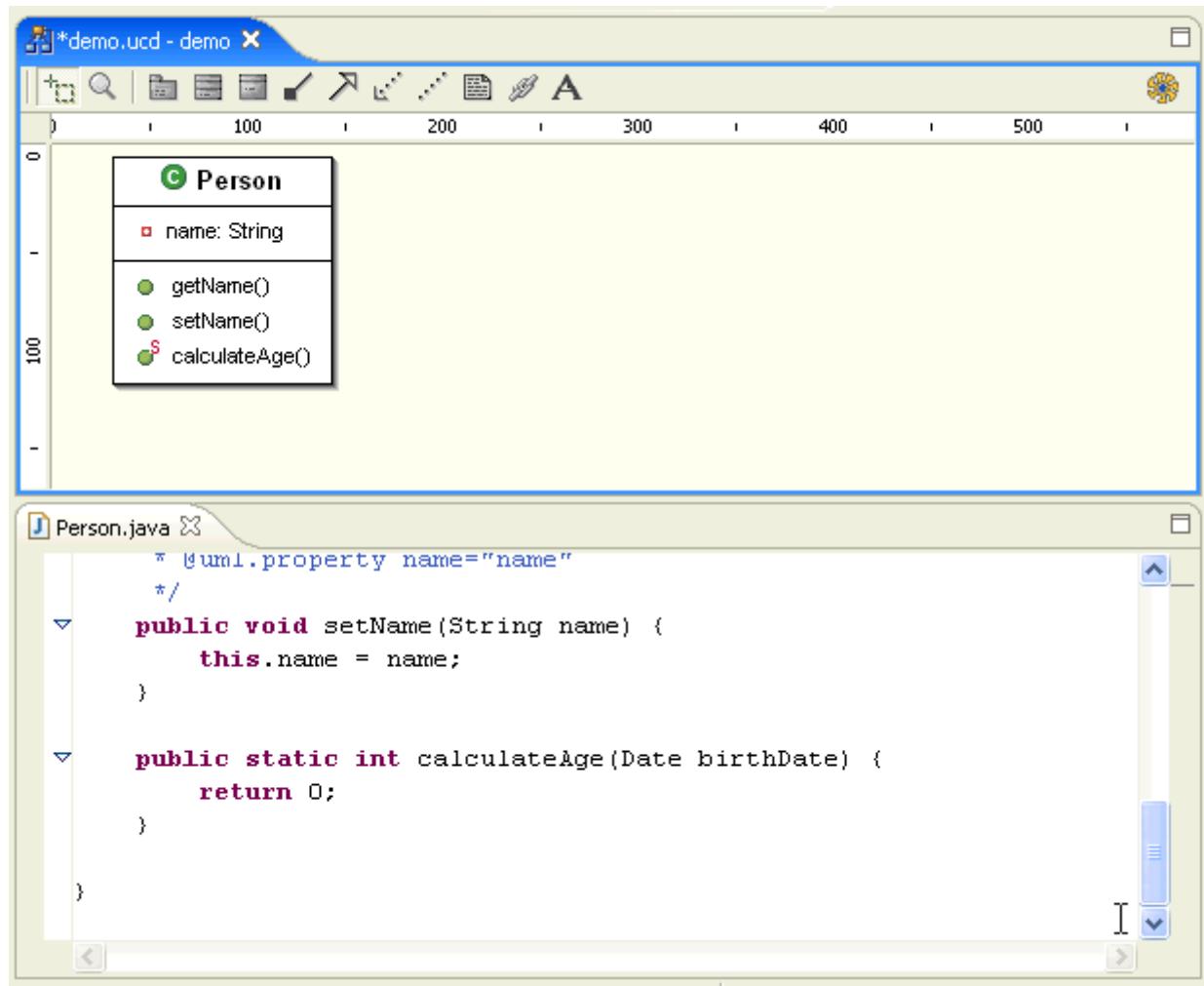
Then you can validate the parameter...



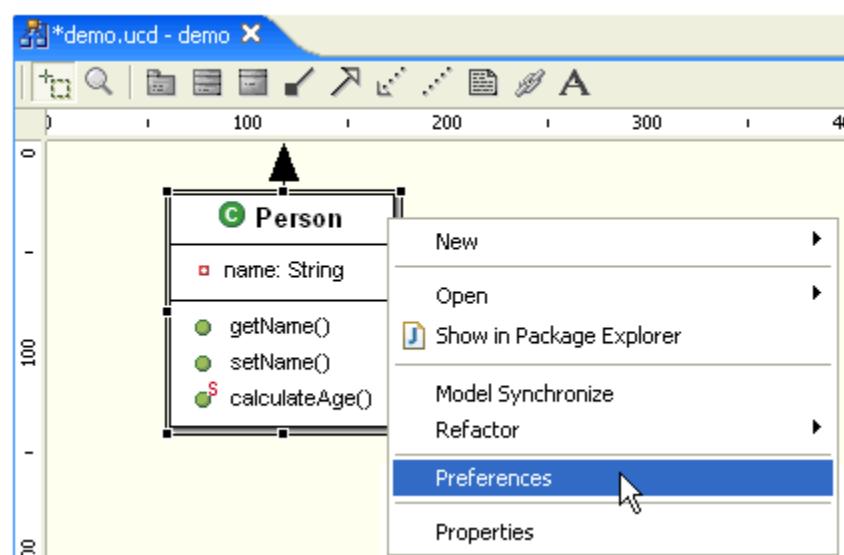
...and the complete method definition.



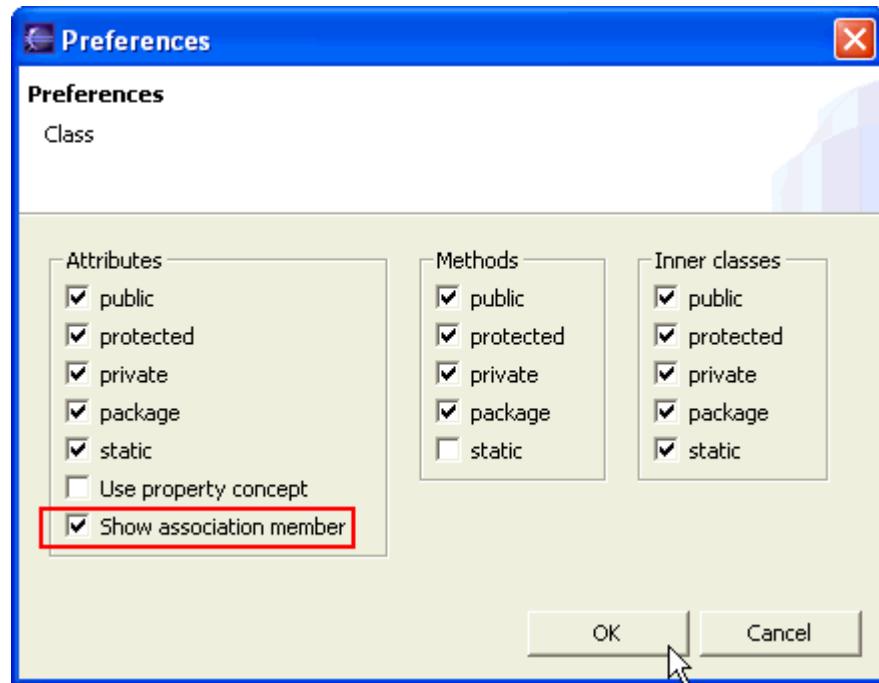
The method appears immediately in the class. If you click on the method, its code appears in the java editor area.



To show attributes and accessors related to associations, select the class in the diagram editor > open the **popup menu** > **Preferences**



Select the "Show association member" checkbox.

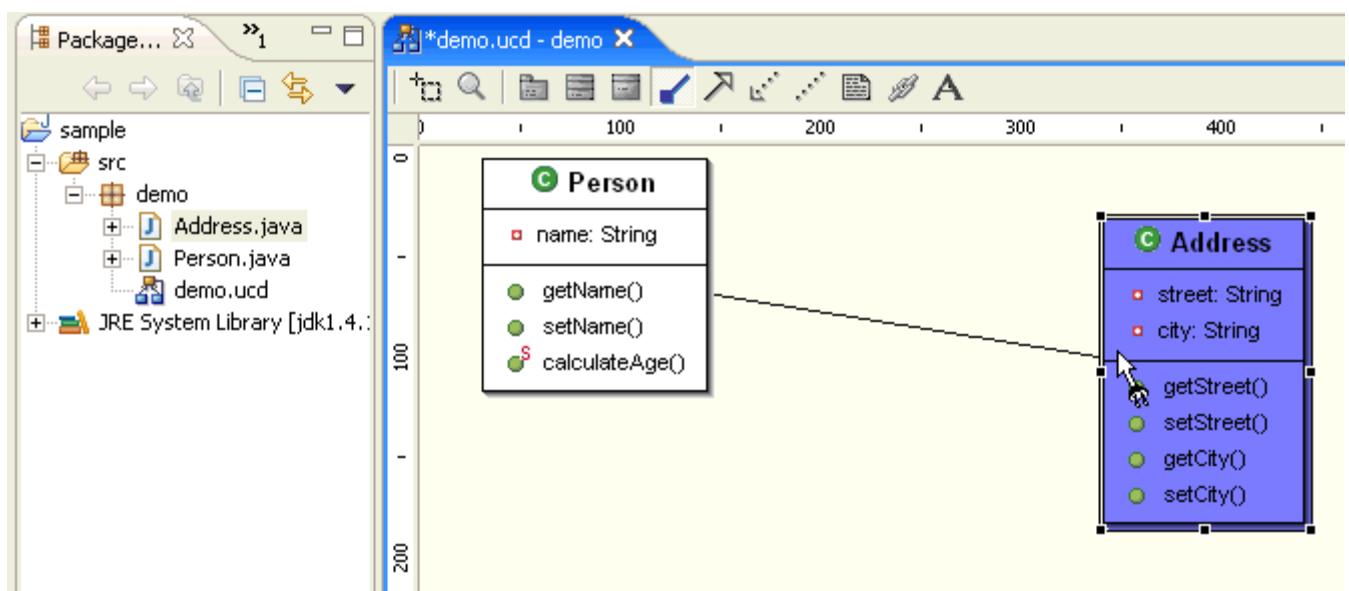


Your diagram now shows new associations attributes and accessors for this class even if the associations are displayed.

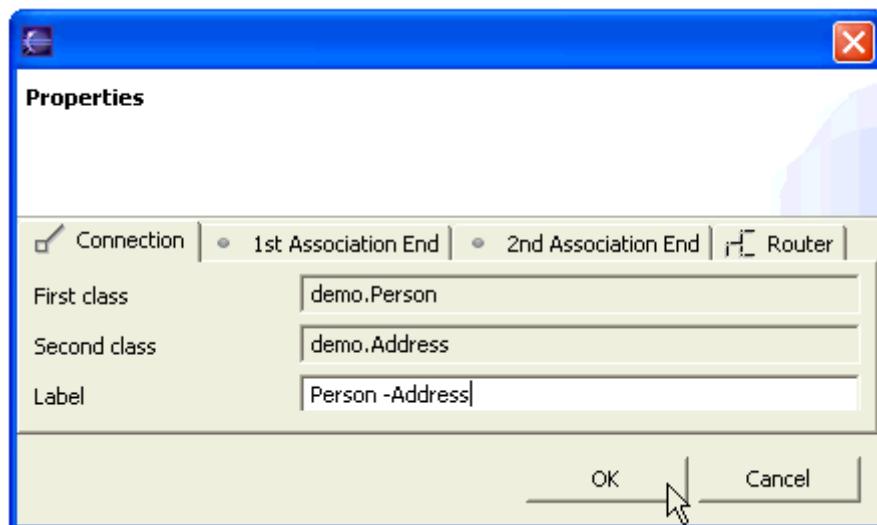


Associations

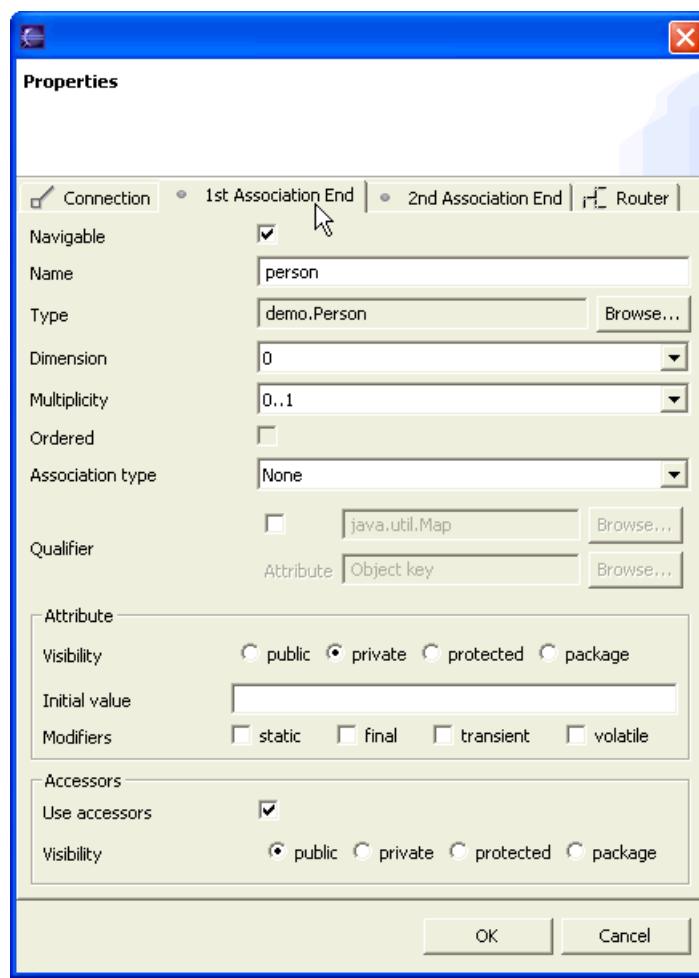
To create an association, select the *Association* toolbar button then click in the original class.



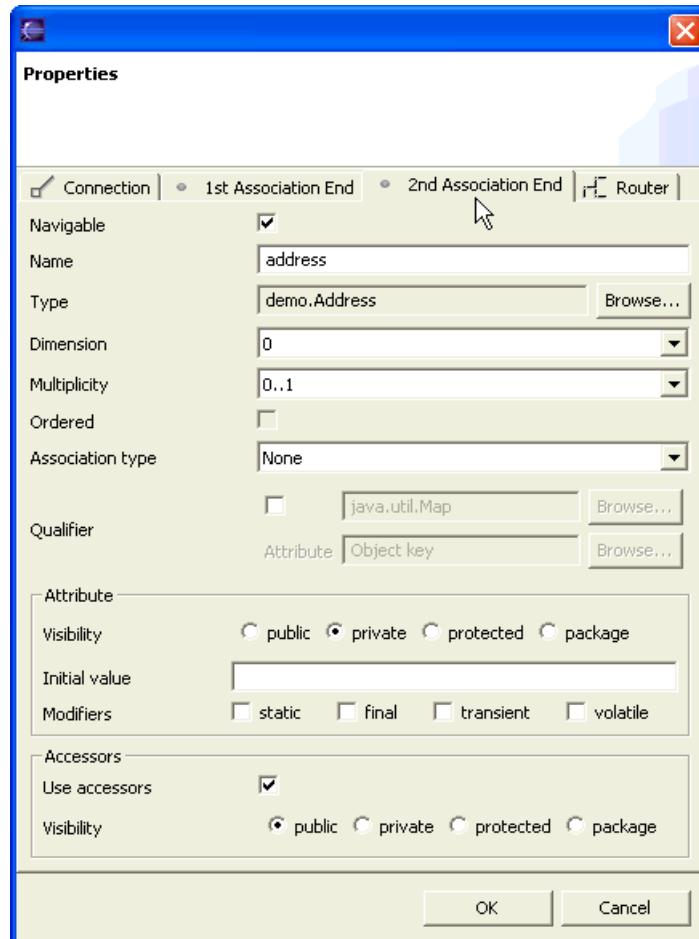
When you click in the target class, a dialog box appears.



Use it to define the first association end...



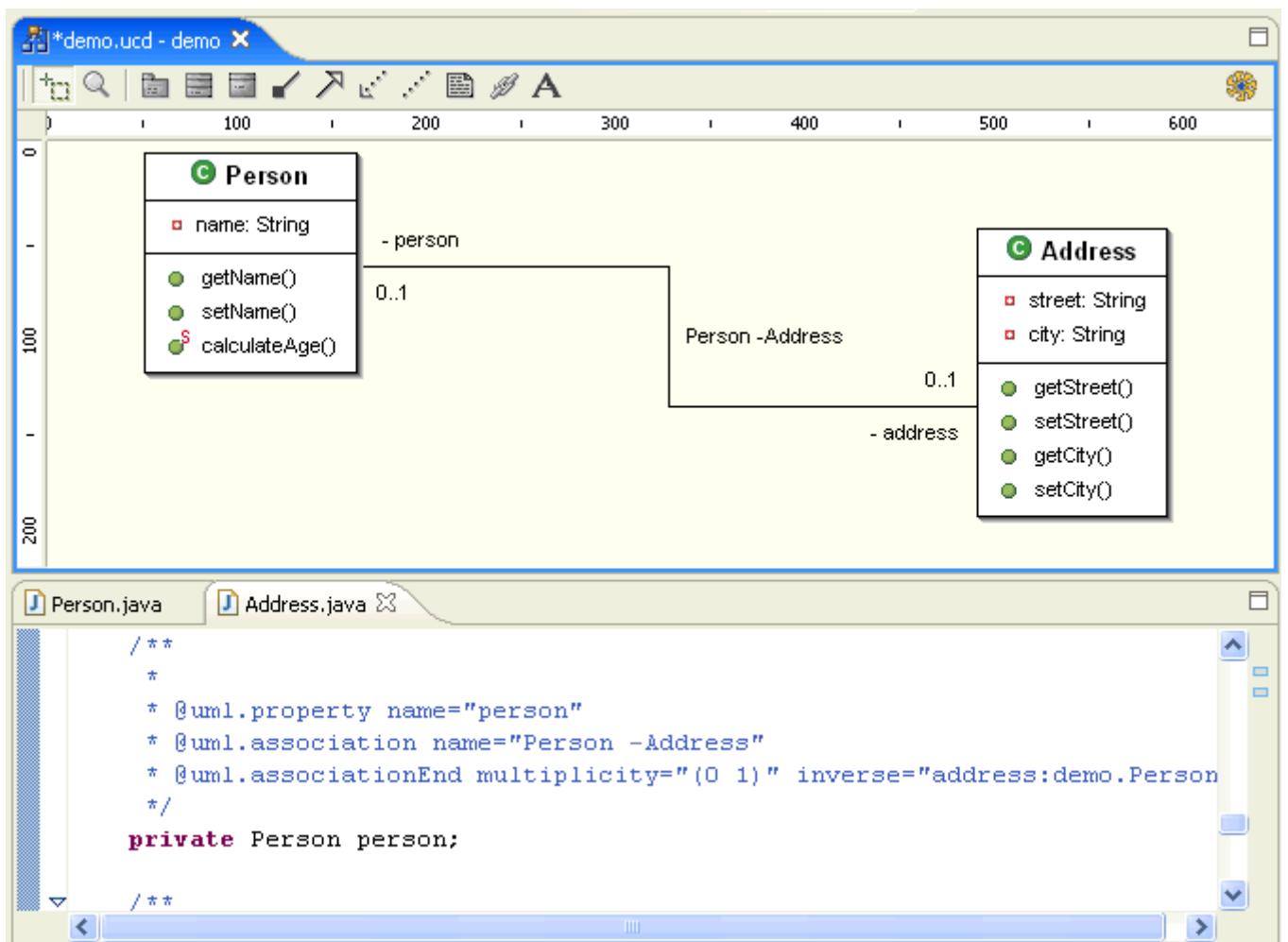
...then the second one...



And finally, the router algorithm and anchor strategy.

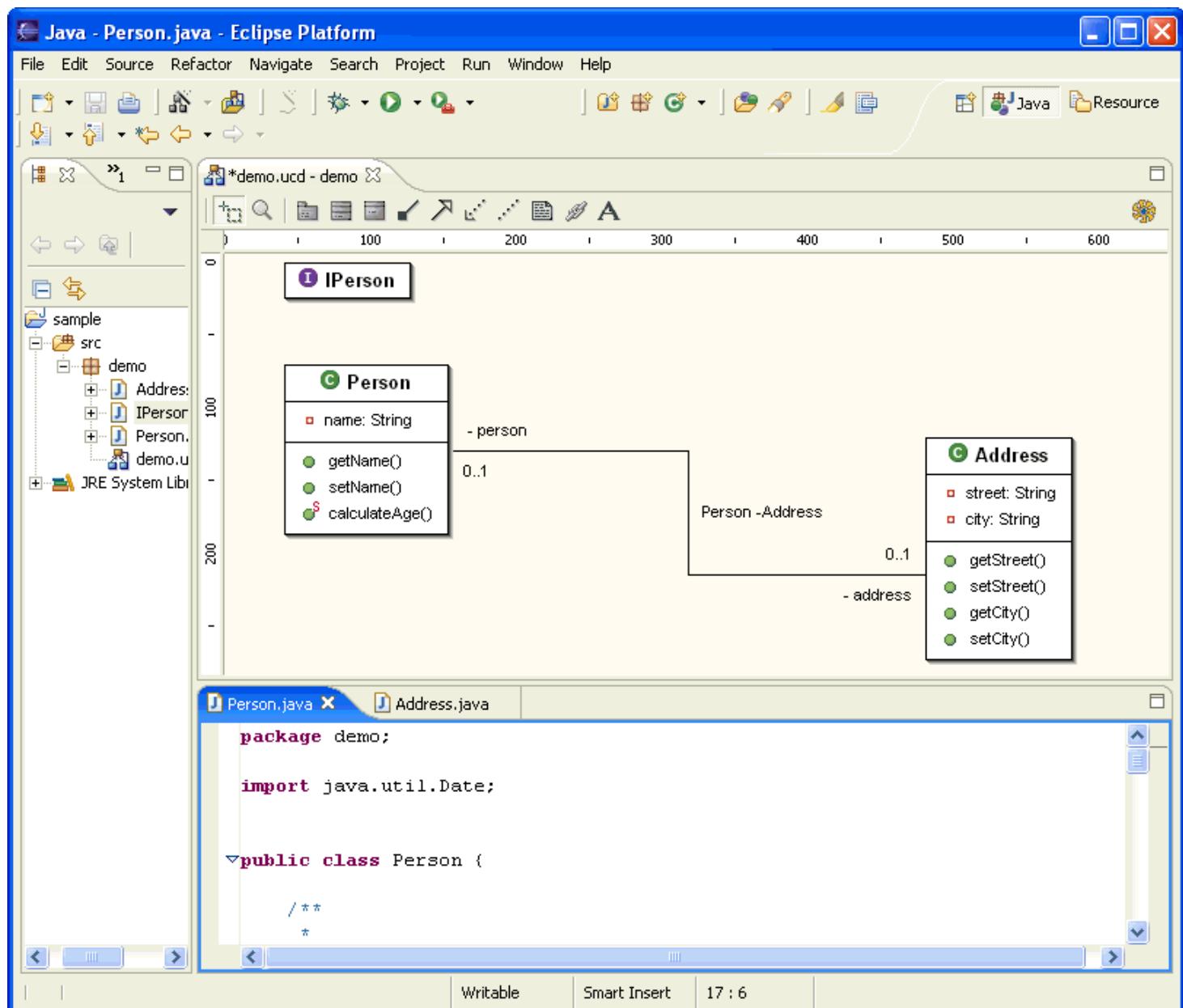


After validation, the association appears in the model.



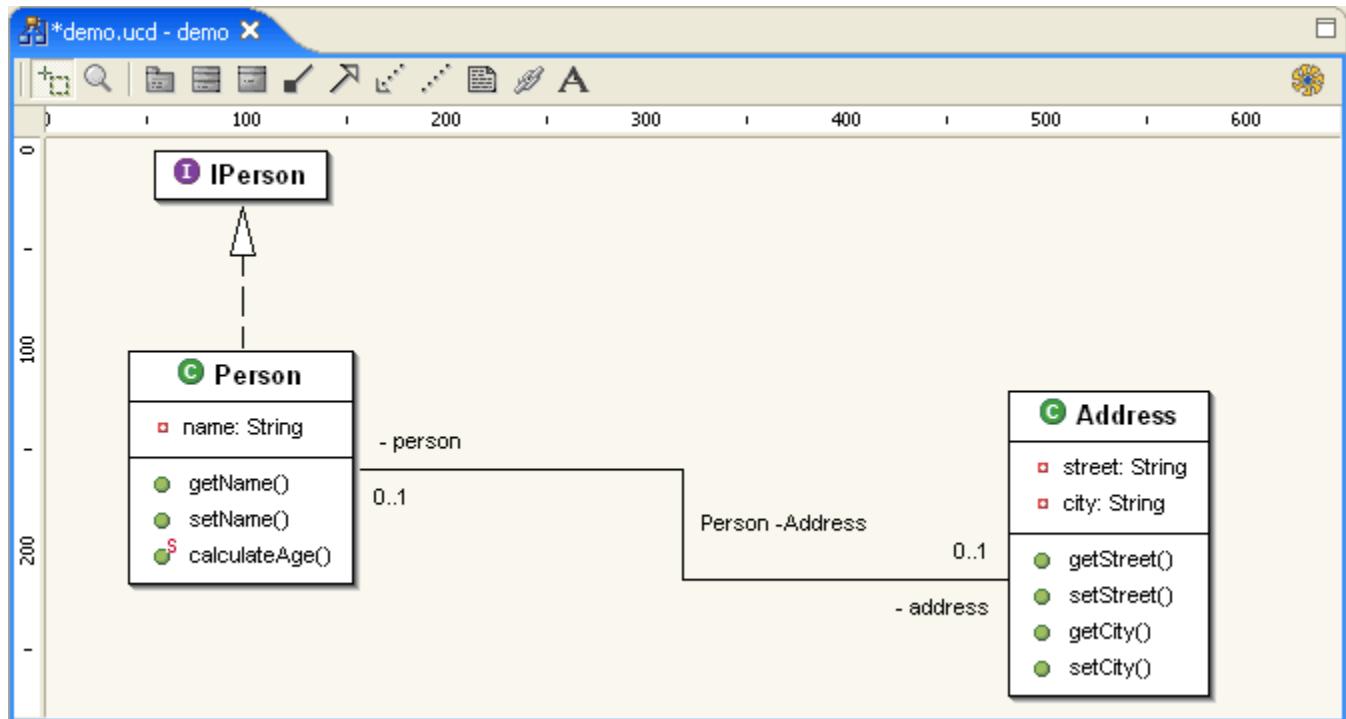
Implementation and Inheritance

To declare an implementation or inheritance link, select the generalization toolbar button.



Click in the source class or interface...

Then release the mouse in the target class or interface. The system will automatically recognize if the link must be an implementation or an inheritance.



Working With Diagrams

The following areas are covered:

- [Rearranging Diagrams](#)
- [Wire](#)
- [Zoom](#)
- [Print](#)
- [Export As Image](#)
- [Customize Perspective](#)
- [Notes](#)
- [Labels](#)
- [Links](#)
- [Key Bindings](#)

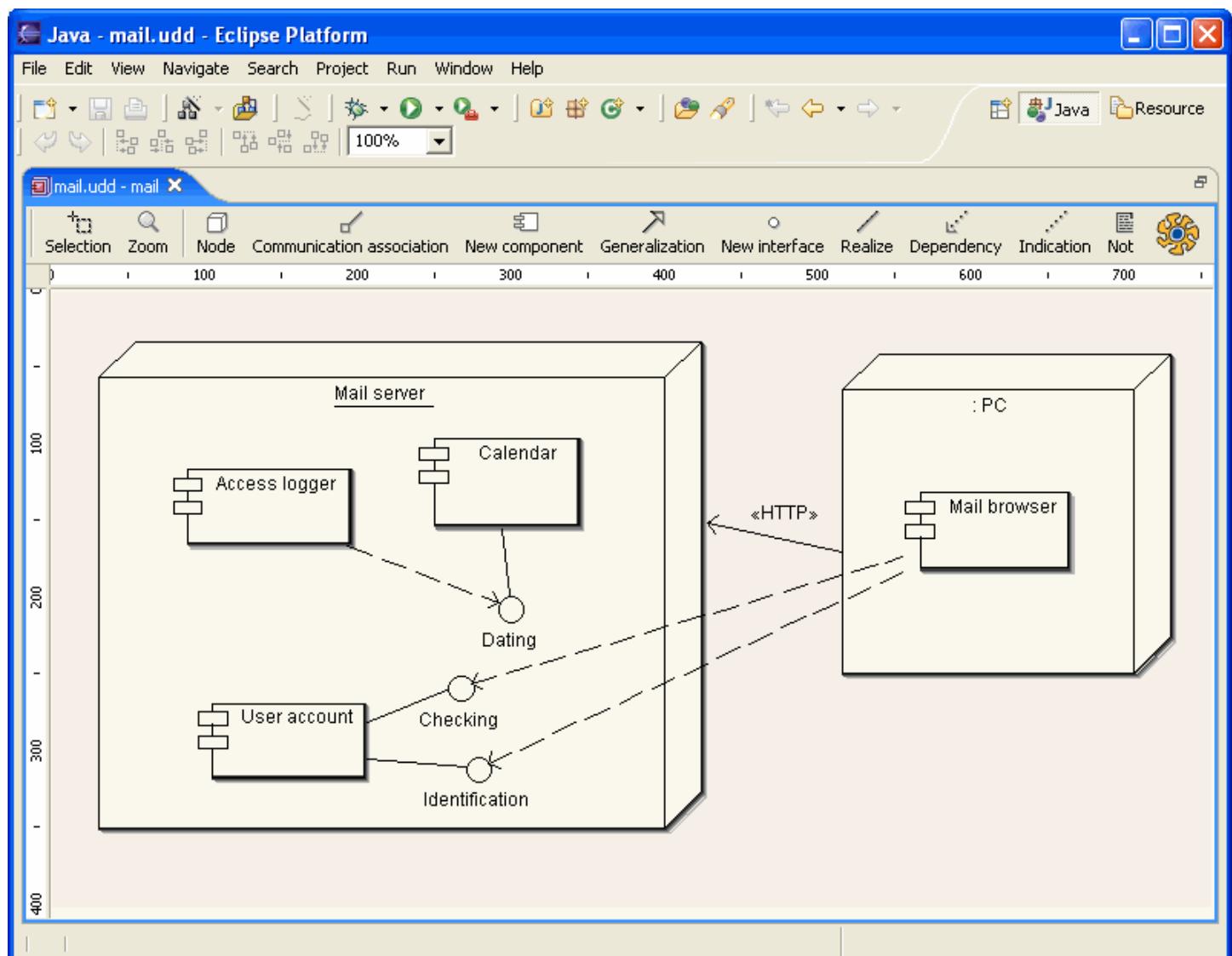
Rearranging Diagrams

You can work on your diagram to adapt the organization using the following elements:

- [Full Screen View](#)
- [Layout](#)
- [Moving elements](#)
- [Routers and Anchors](#)
- [Alignment](#)

1. Full Screen View

It is possible to work on a full screen diagram editor by double-clicking the diagram file tab. The following example was created by double clicking on the com.udd deployment diagram, then selecting the zoom icon on the toolbar and finally double clicking inside the deployment diagram editor.



2. Layout

EclipseUML allows you to use layout functionalities. We recommend using Layout for a class diagram reverse and then manually reorganizing your diagram.

The layout will reorganize your diagram using mathematical algorithm.

To open the layout function, open the diagram editor's popup menu and select layout.

3. Moving elements

Advanced multiple selections are proposed, as well as group/ungroup and key shortcuts for more precise functions.

Selection of an element:

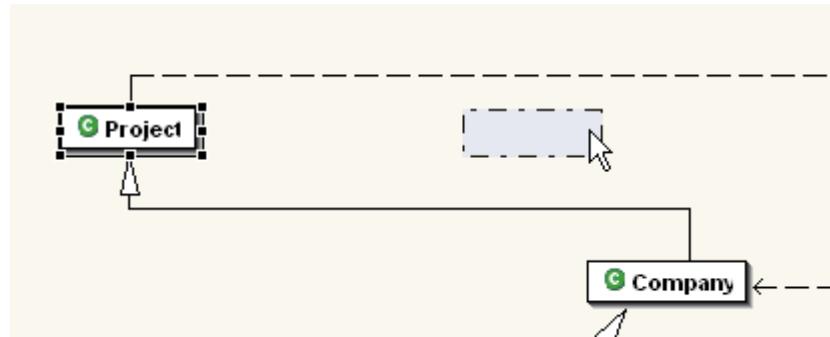
EclipseUML allows you to move every element of the diagram using the mouse or the keyboard.

Moving is possible with:

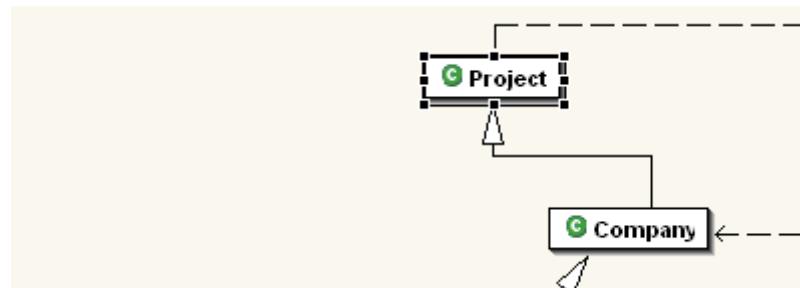
Mouse: Left click and keep your finger on the button

Keyboard: Ctrl+Alt+arrows

The following example shows how to move the Project class using the mouse:



When the mouse button is released, the Project class has moved.

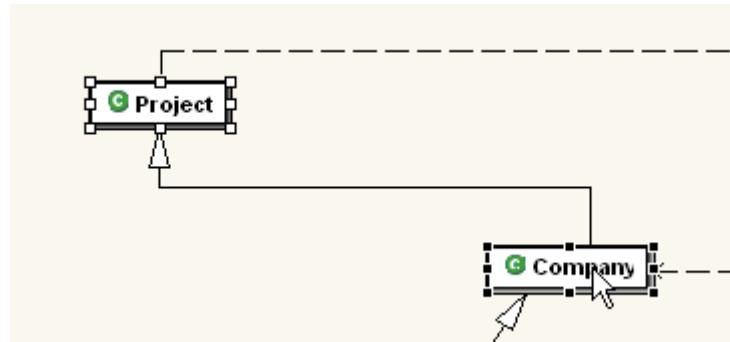


Selection of a group of elements:

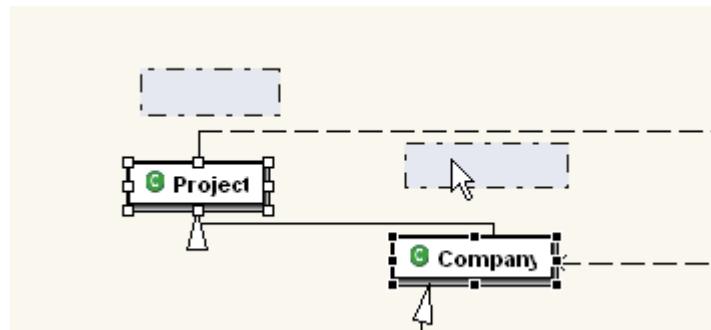
Moving is possible with:

1. Mouse: Shift+left click on each element then move using the mouse.
2. Keyboard: shift+arrows then move using the arrows (if you press Ctrl+arrows, the element will not be selected so use shift+arrows to select the element).
3. Click on selection in the tool bar and crop elements.

The following example shows how to move Project and Company classes using the mouse:



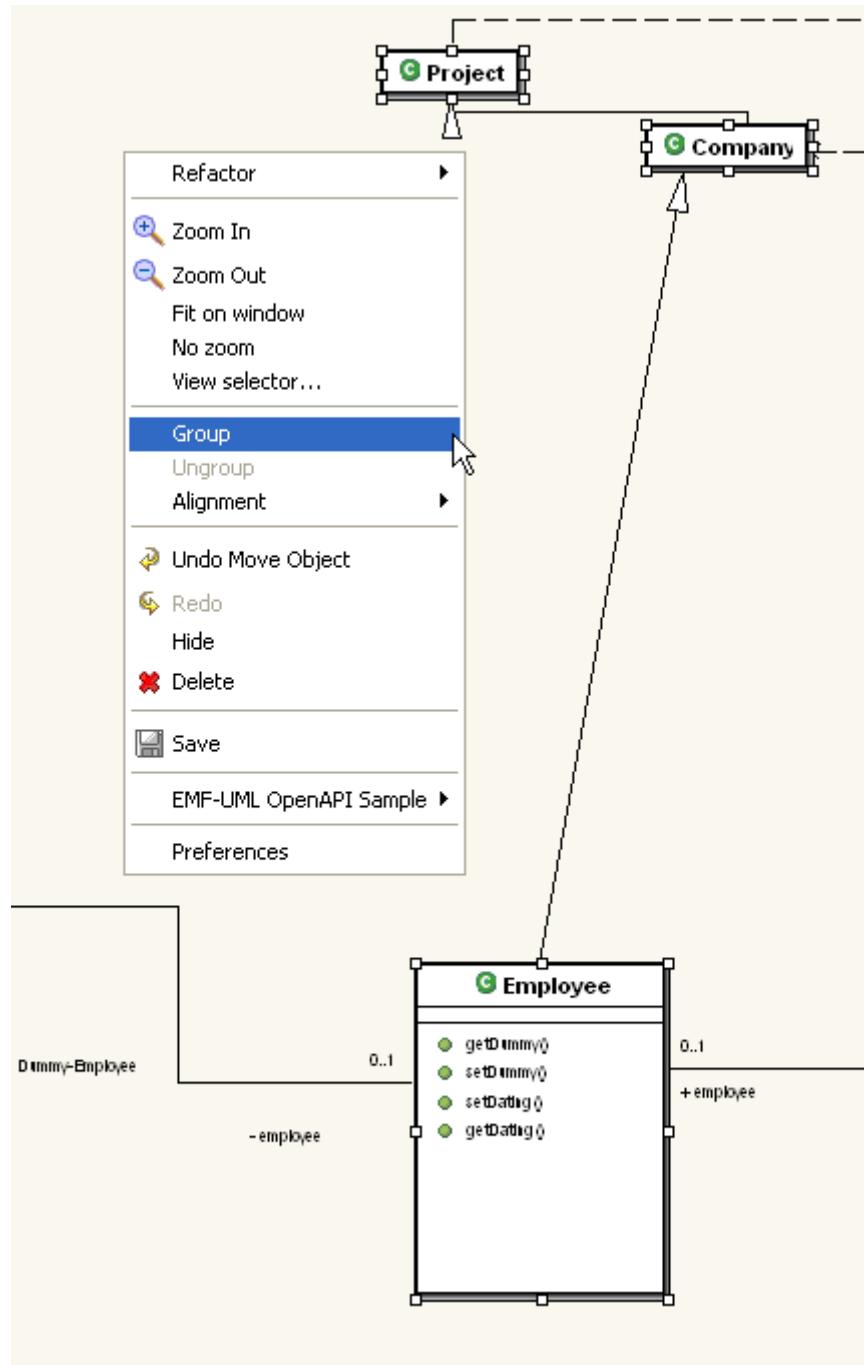
When the mouse button is released, Project and Company classes have moved.



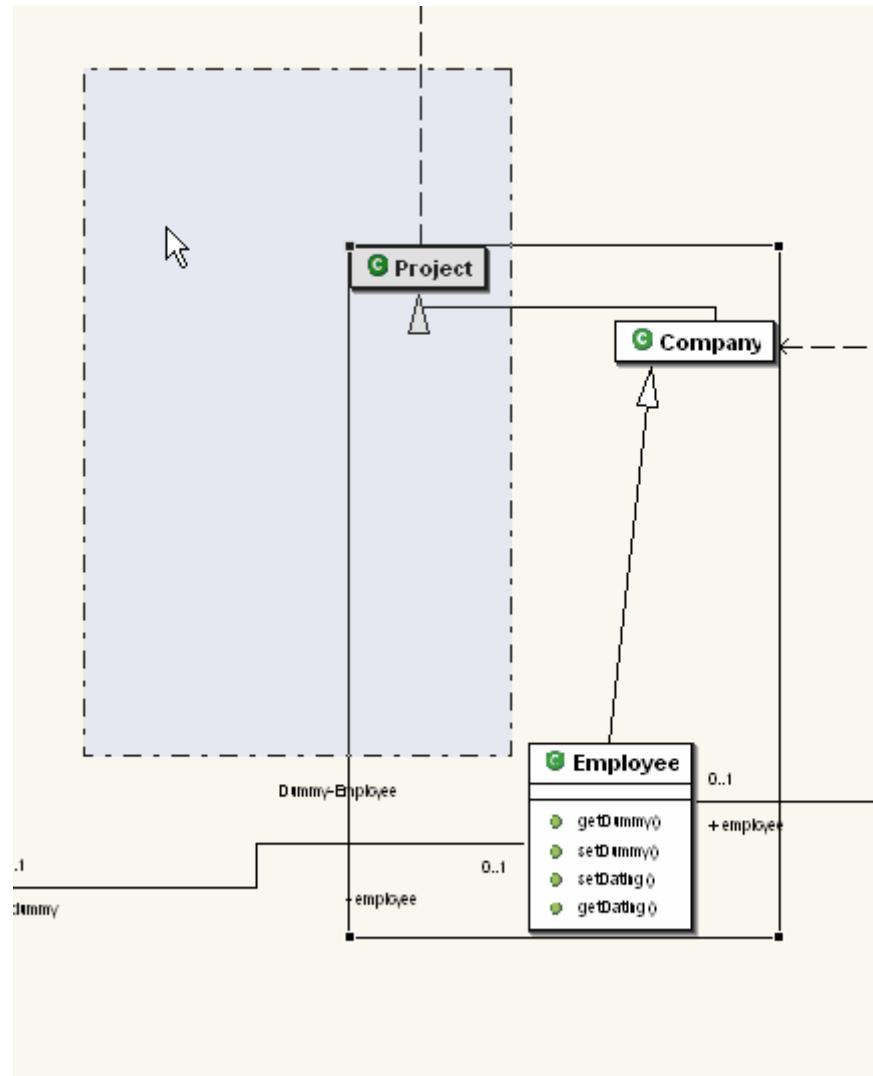
Selection of a group of elements and the use of the group function.

EclipseUML allows you to select a group of elements and move the selected elements.

In the following example we have selected Project, Company and Employee classes and opened the pop up menu>group



The square is blue, because it is moving and the button of the mouse is still pressed. It represents the group composed by the three elements (Project, Company and Employee classes).

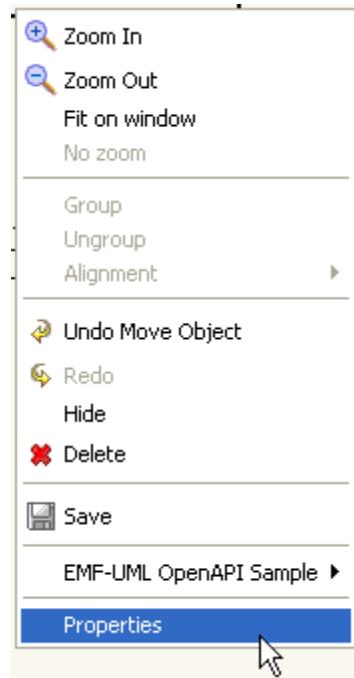


The group classes will always move together.
Ungroup will deselect the links between elements.

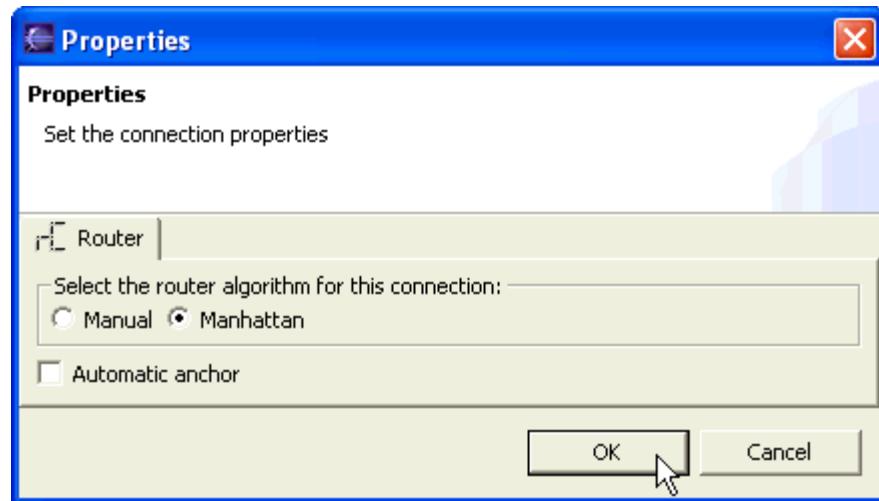
4. Routers and anchors

You can customize every link inside your diagram editor.

Right Click on a link between two elements >open the pop up menu>Properties

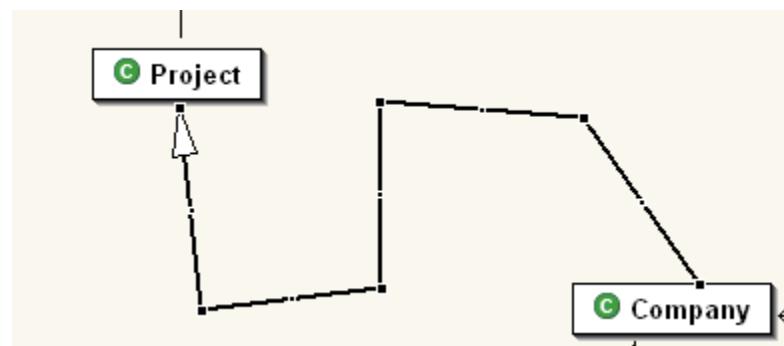


You can select the router algorithm and Anchor for each link



Router:

Manual allows to move the extremity of each link.

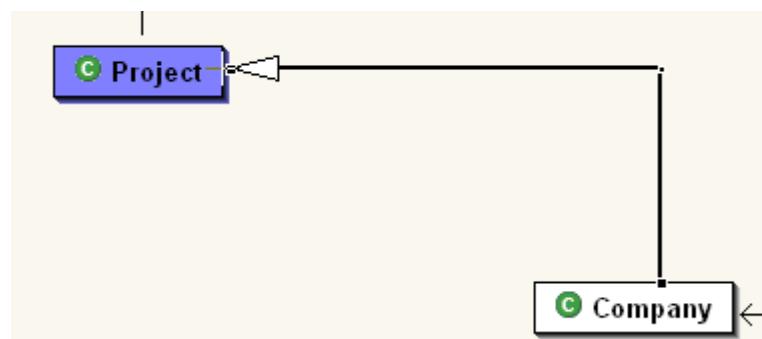


Manhattan uses automatic functions to design the link.

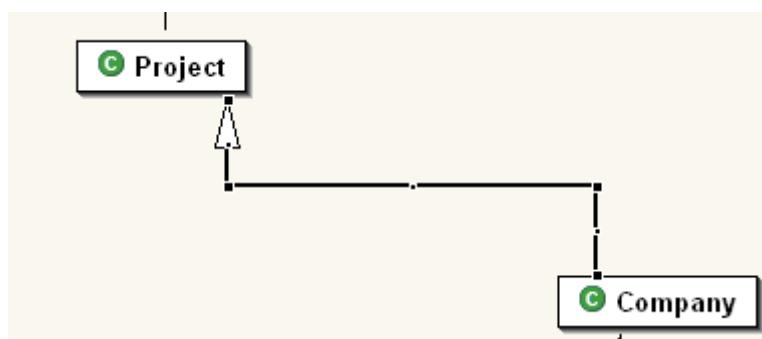


Anchor:

Unselecting Automatic anchor allows to fix the extremity anywhere. Move the anchor inside the element which will become blue when the anchor is properly fixed.



Automatic anchor uses algorithm to fix the anchor automatically (the anchor cannot be moved)



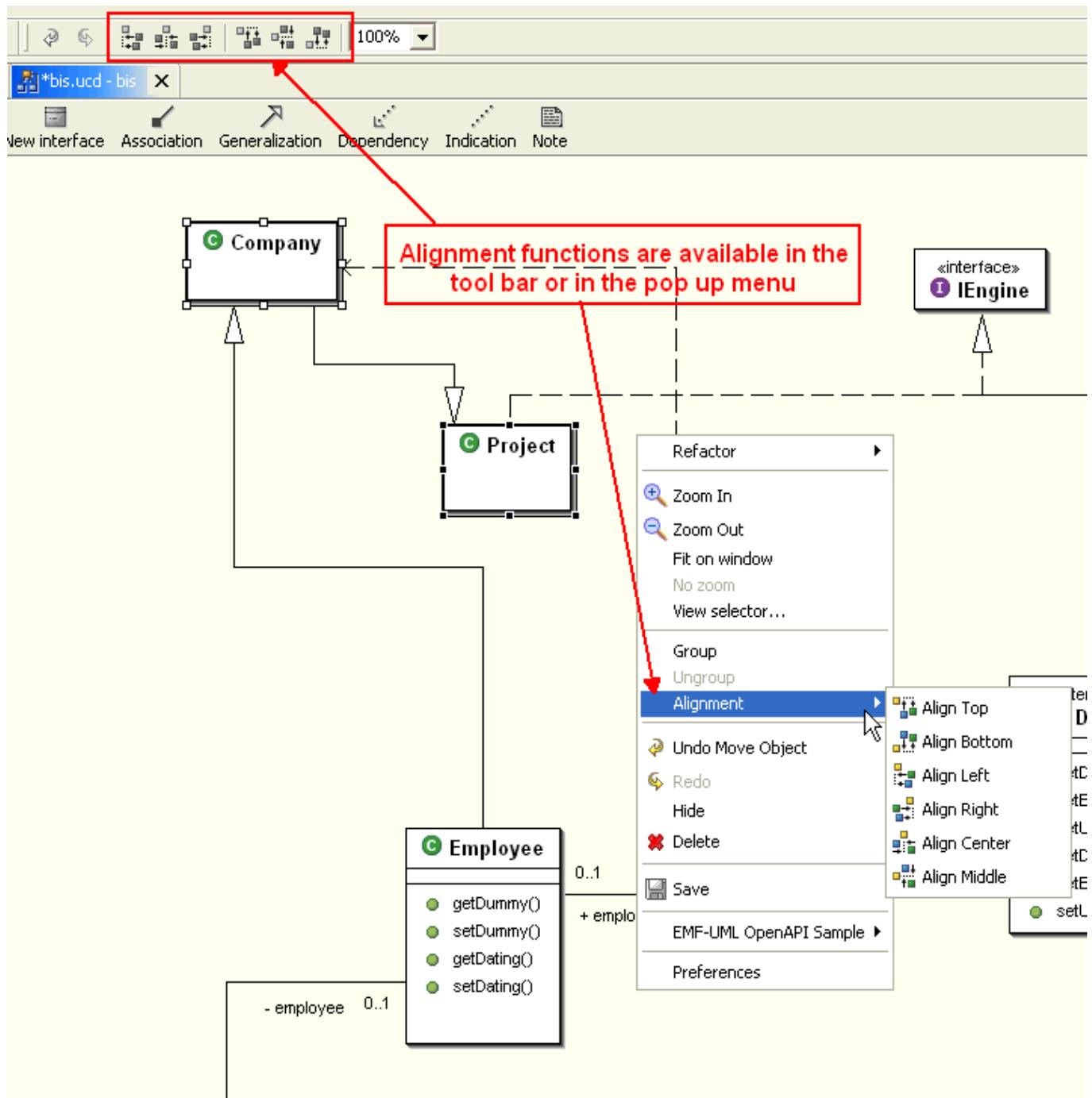
5. Alignment

EclipseUML allows you to select a group of elements and to arrange the alignment. Six alignments are possible:

1. Align Top
2. Align Bottom
3. Align Left
4. Align Right
5. Align Center
6. Align Middle

The following example shows where you can select the alignment function:
Select the alignment function from:

1. The workbench window's toolbar
2. The diagram editor's popup menu



Wires

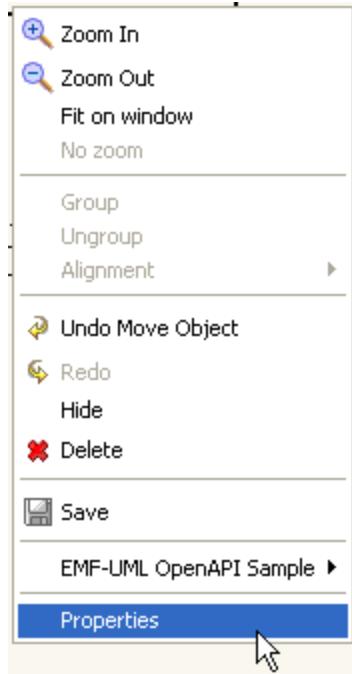
You can work on your diagram to adapt the organization using the following elements:

1. [Routers and Anchors](#)
2. [Wire colors](#)

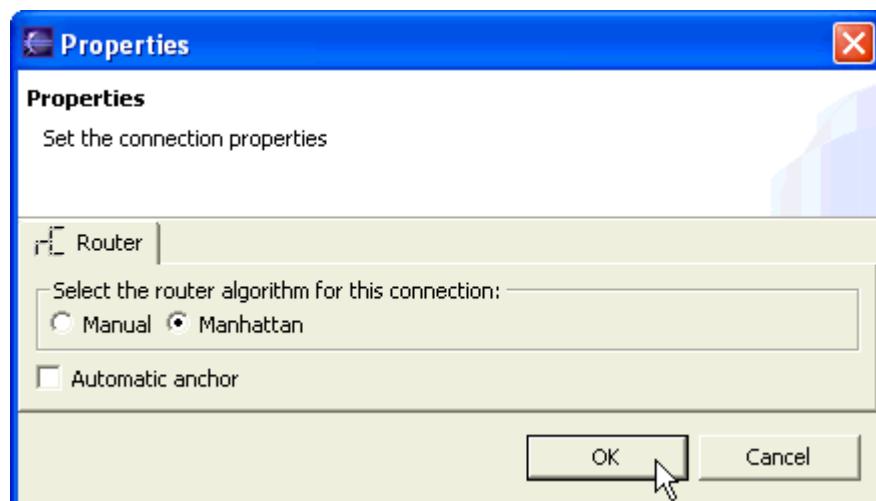
1. Routers and anchors

You can customize every link inside your diagram editor.

Right Click on a link between two elements->open the pop up menu->Properties

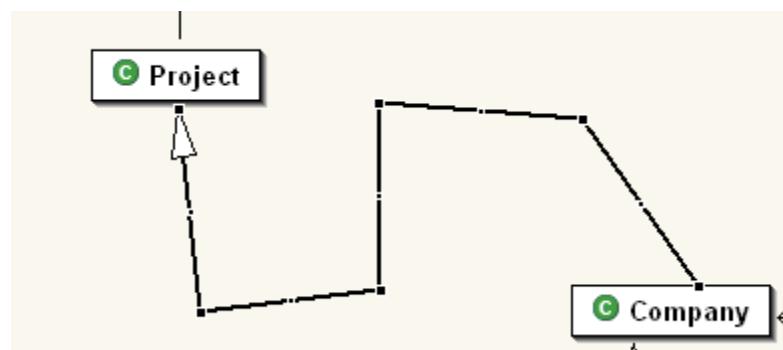


You can select the router algorithm and Anchor for each link



Router:

Manual allows to move the extremity of each link.

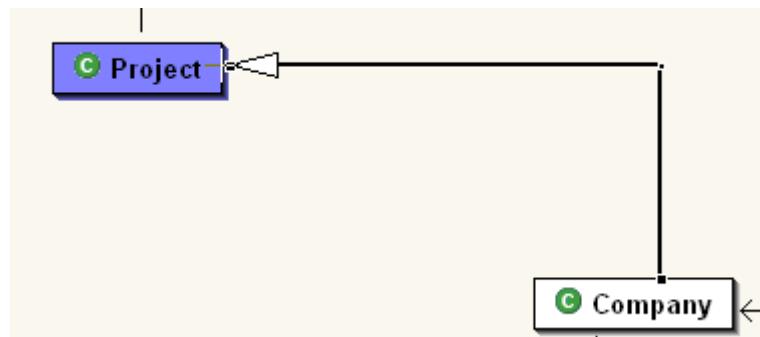


Manhattan uses automatic functions to design the link.

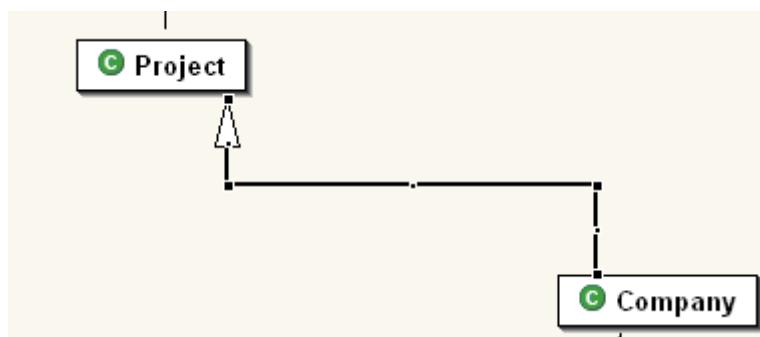


Anchor:

Unselecting Automatic anchor allows to fix the extremity anywhere. Move the anchor inside the element which will become blue when the anchor is properly fixed.



Automatic anchor uses algorithm to fix the anchor automatically (the anchor cannot be moved).



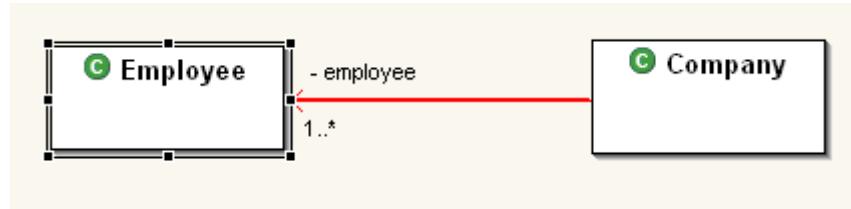
2. Wire colors

Selecting any element of a diagram will automatically colorize its related wires. In complex diagrams, this feature allows a better visibility of the immediate environment of a selected element.

Colors are different depending on the wire direction.

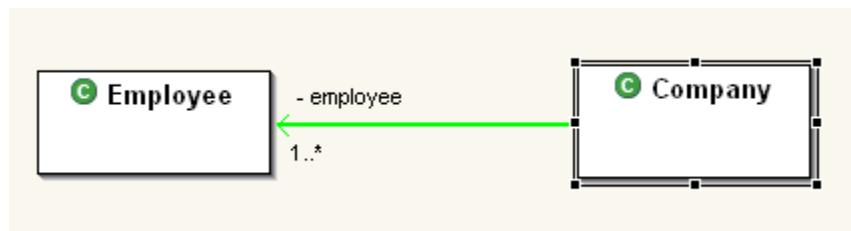
2.1 Incoming wires

All incoming wires will be displayed in red when selecting an element.



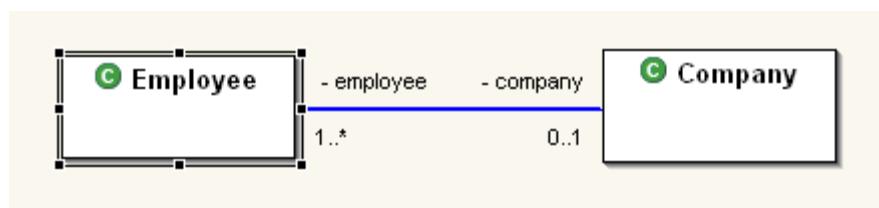
2.2 Outgoing wires

All outgoing wires will be displayed in green when selecting an element.



2.3 Bi-directional wires

Bi-directional wires will be displayed in blue when selecting an element.



Zoom

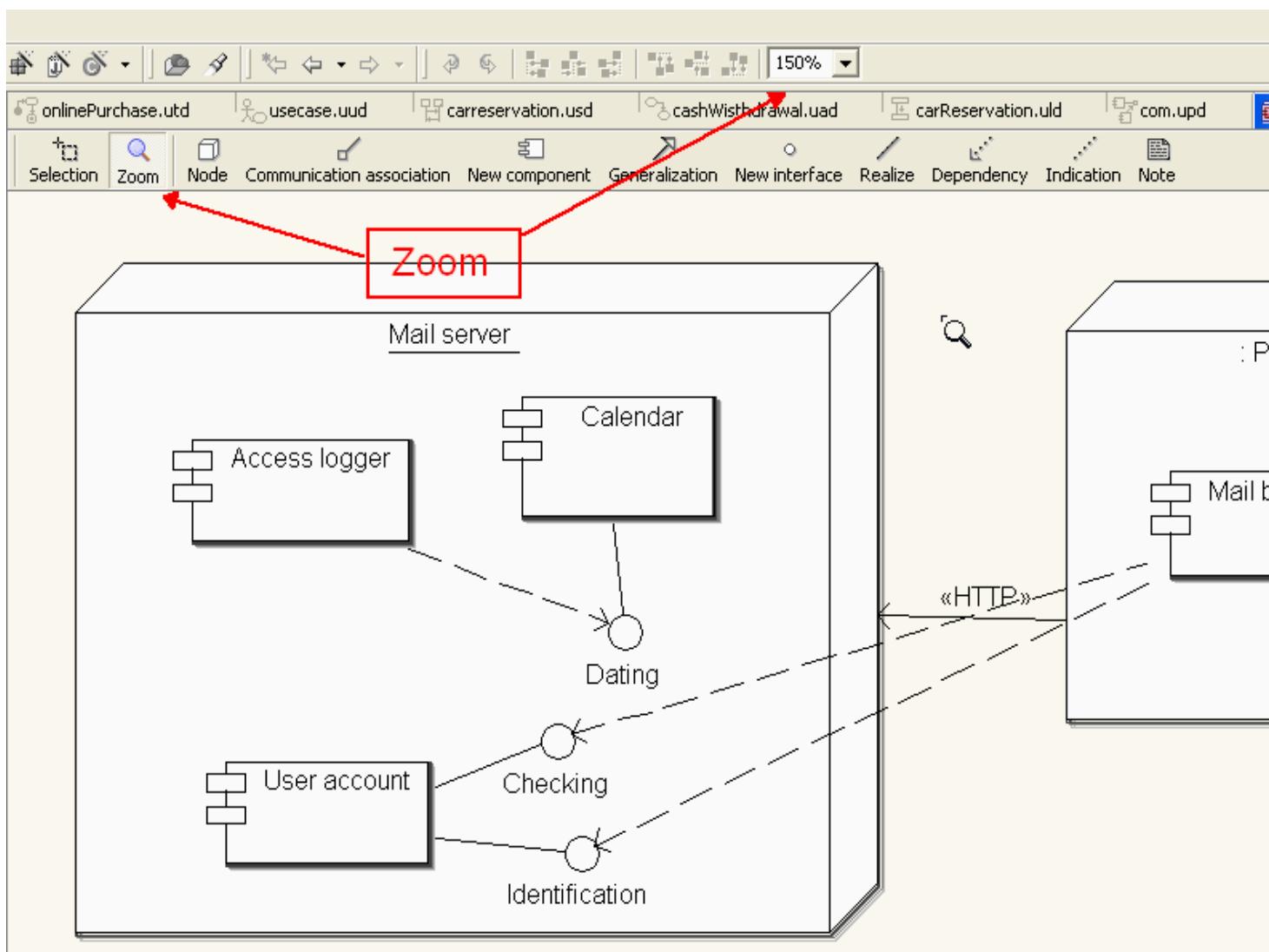
EclipseUML allows you to zoom in the diagram editor. By **selecting the zoom mode**, you can work in a more efficient way with your diagrams and be focused on your diagram, not only on your Java code.

Zooming is possible by selecting:

- the zoom icon in the toolbar
- the zoom percentage in the Workbench window's toolbar
- the diagram editor's popup menu

Three major functions are available within the Zoom function:

1. [Zoom in and out](#)
2. [Overview](#)
3. [Cropping](#)



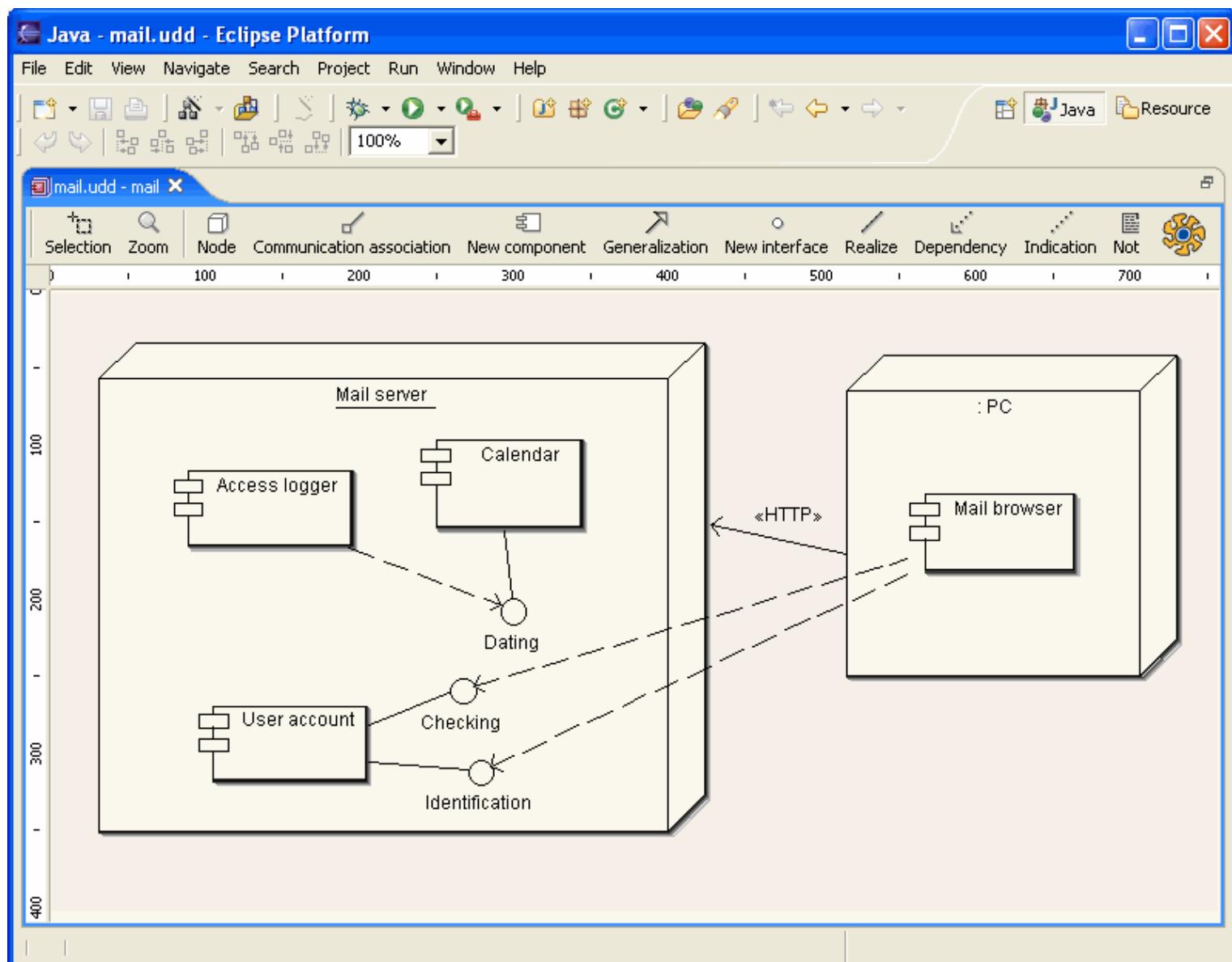
1. Zoom in, Zoom out

Selecting the **zoom icon** in the toolbar allows you to use the following zoom functions:

1. zoom in > right click
2. zoom out > left-click

2. Overview

You can see a complete **overview** of your diagram by selecting the zoom icon in the tool bar and **double clicking inside the diagram editor**.



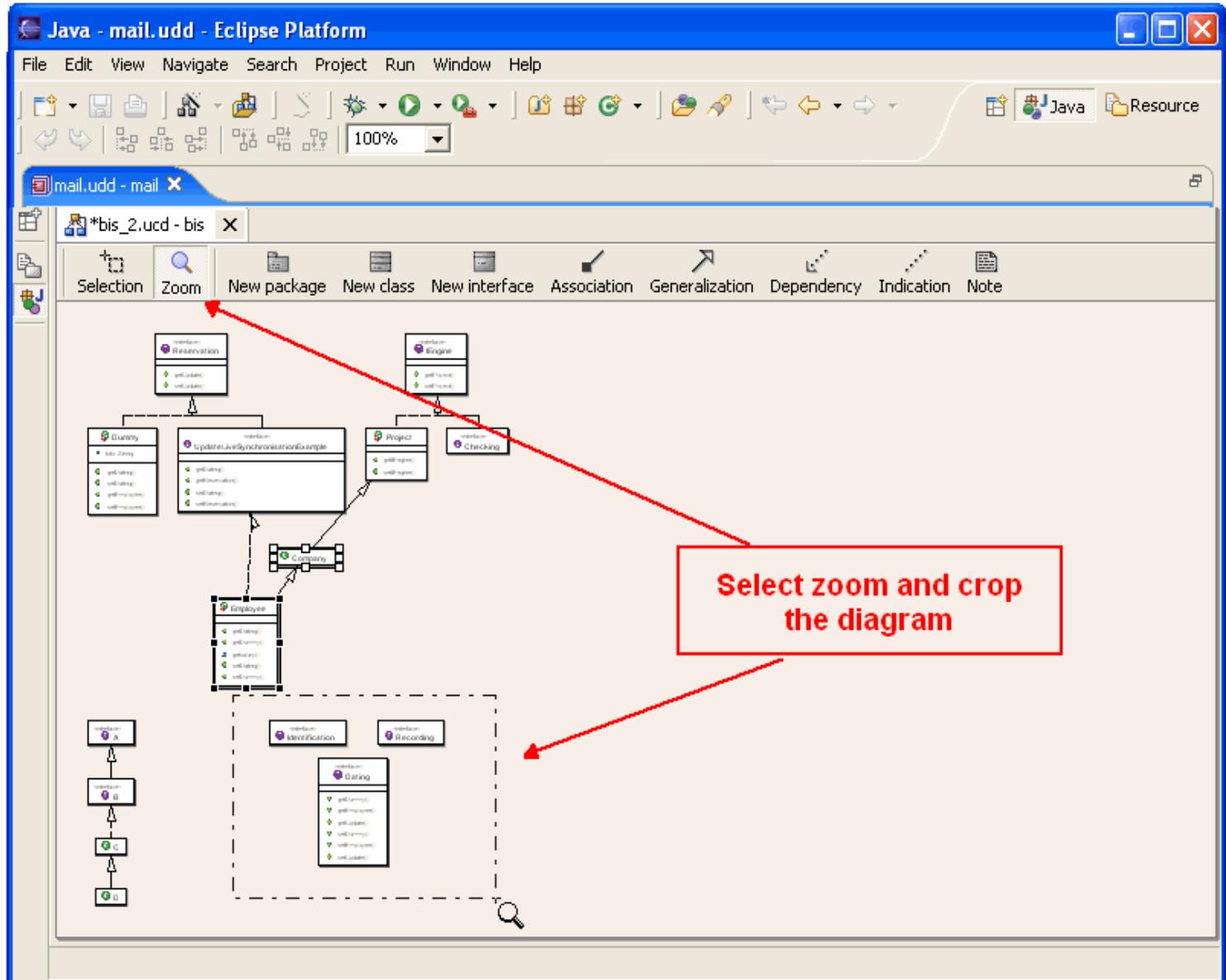
Another double-click brings you back to the previous state.

3. Cropping

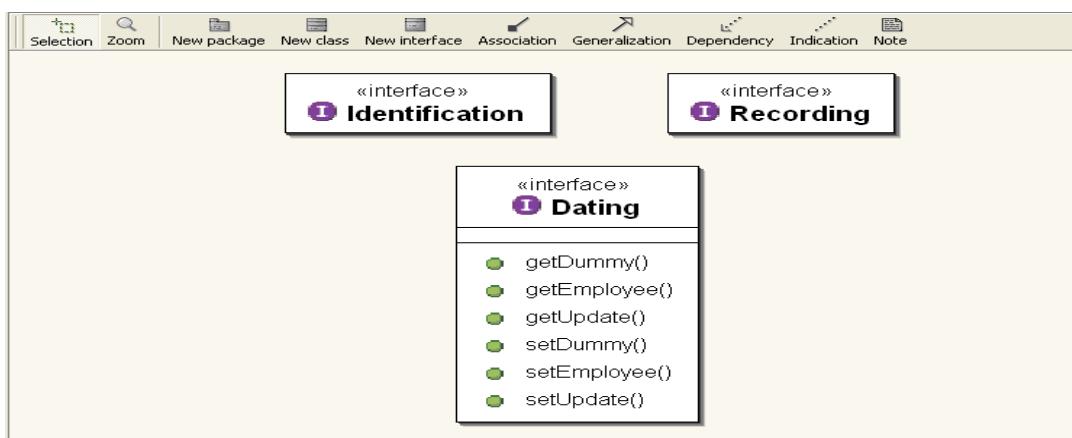
You can select a part of the diagram that you want to zoom in.
Select the Zoom icon in the toolbar and crop inside the diagram editor.

In the following example, we have an overview of a class diagram. We need to zoom inside this diagram and decide to crop.

Zoom in the toolbar must be activated, then use the mouse to crop.



As soon as the mouse button is released, the selected area is enlarged.



Print Options

The print function is available from:

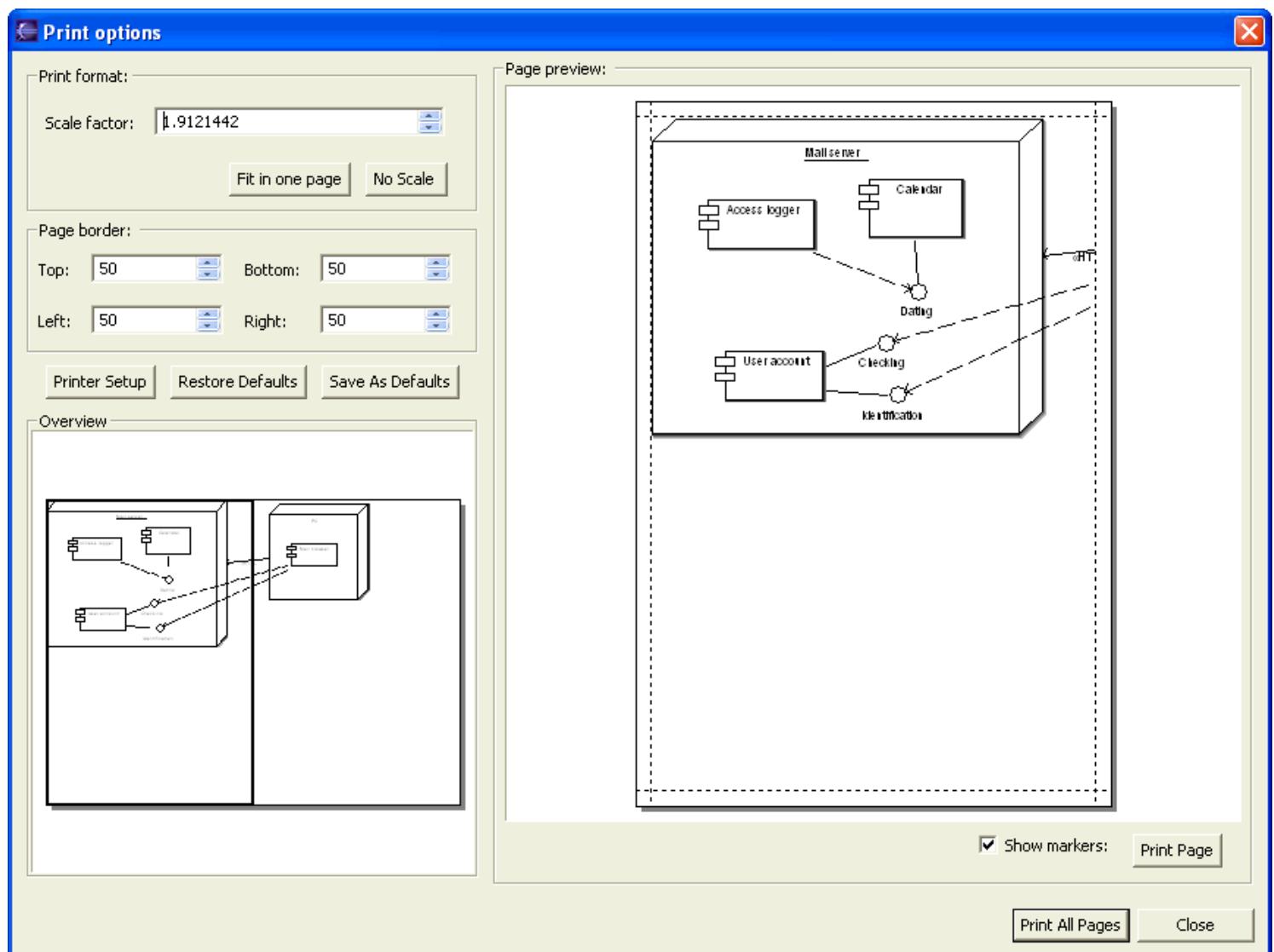
- The diagram editor's popup menu > print (to open the popup menu, right click inside the diagram editor and be sure not to select any element).
- From the menu bar select **File > Print**

Print option allows you to choose four different parameters:

1. **Print format** : adapt the scale factor and have a look at the page preview to see the result. Fit on one page will automatically select the appropriate scale.
2. **Page border** : indicates the border customization of your diagram editor image and its export to your printer.
3. **Overview** : shows all the printing pages of the document. Selecting a page in the overview will automatically select this page in the Page preview window.
4. **Page preview** : shows the page before it is printed.

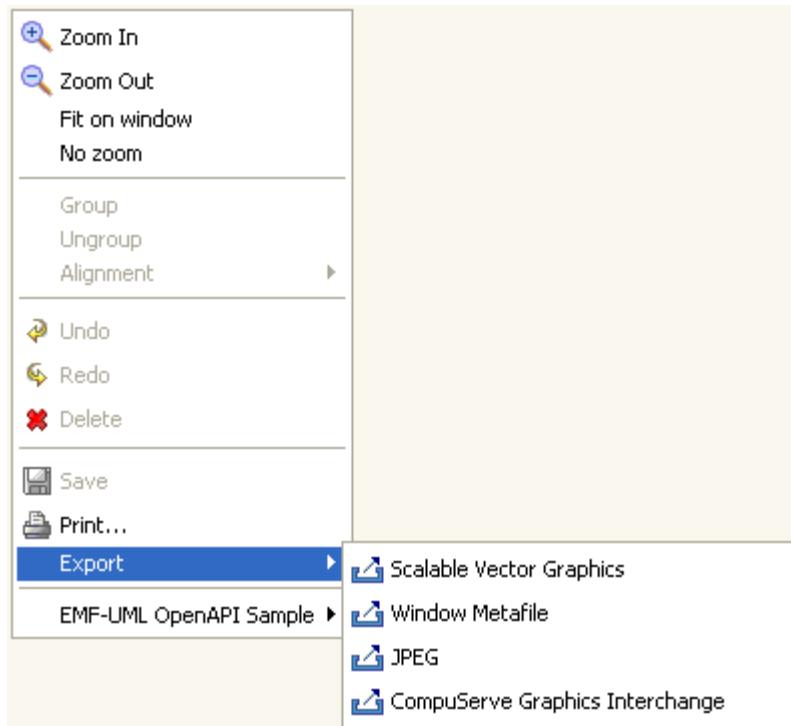
Diagrams can be printed using Print option or keeping the global [Print preferences](#).

The following picture is an example of the Print option window.



Export as Image

Different exporting formats are available. You can access this feature from the diagram editor's popup menu by right-clicking on the background of a diagram.



Several formats are available for export :

- Scalable Vector Graphics (SVG)
- Window Metafile (WMF)
- JPEG
- CompuServe Graphics Interchange (GIF)

Special Characters Support

Diagrams using special characters such as Japanese, Korean, Chinese, etc... need to use an appropriate font. The export font can be configured in the [preferences](#).

Customize Perspective

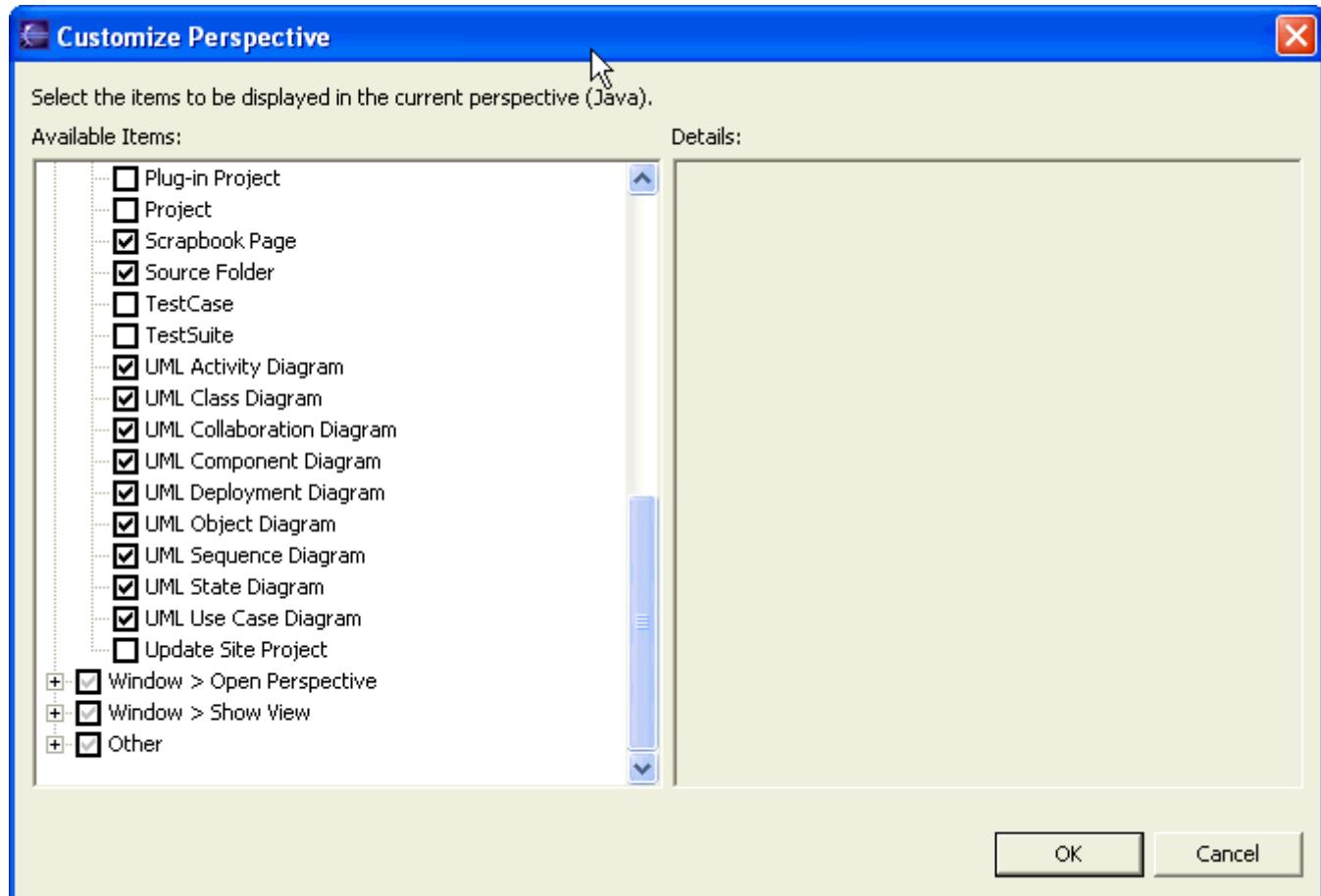
It is important to customize your Eclipse workbench perspectives. Eclipse UML diagrams and functions can be customized using the Customize Perspective.

You can customize your diagram perspective using the following elements:

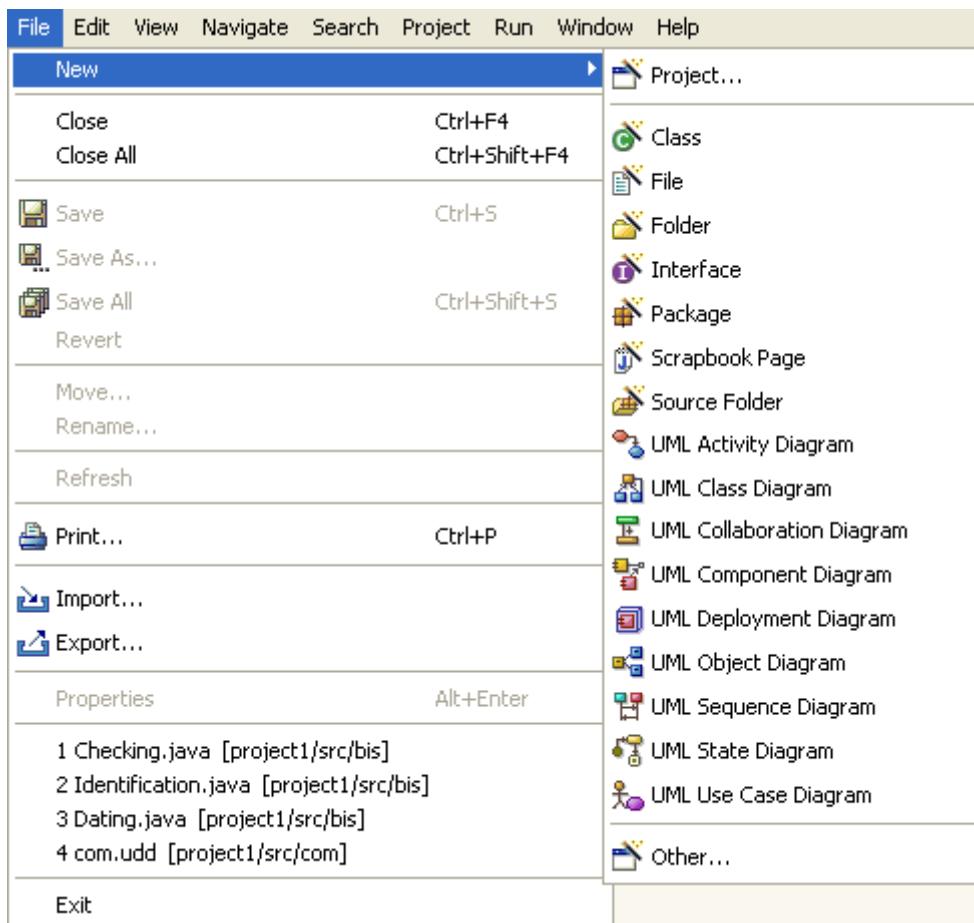
1. [Customize Perspective](#)
2. [Toolbar](#)

1. Customize Perspective

Select in the menu bar **Window > Customize Perspective > File > UML** submenu



Every selected diagram will be available directly from the menu bar: select **File > New** submenu

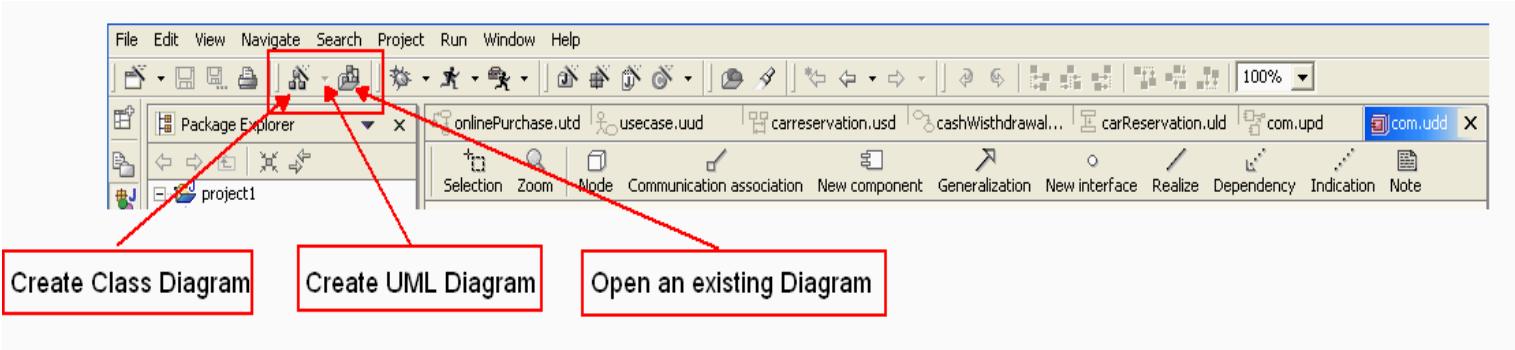


2. Toolbar

The workbench window's toolbar can be customized for UML needs. Select from the menu bar **Window > Customize Perspective > Other > UML**

The three following icons will be shown in the Workbench window's toolbar.

1. Create a class diagram by clicking on the icon
2. Create UML diagrams by activating the drop-down menu (simply click on the down arrow)
3. Open an existing diagram by clicking on the icon



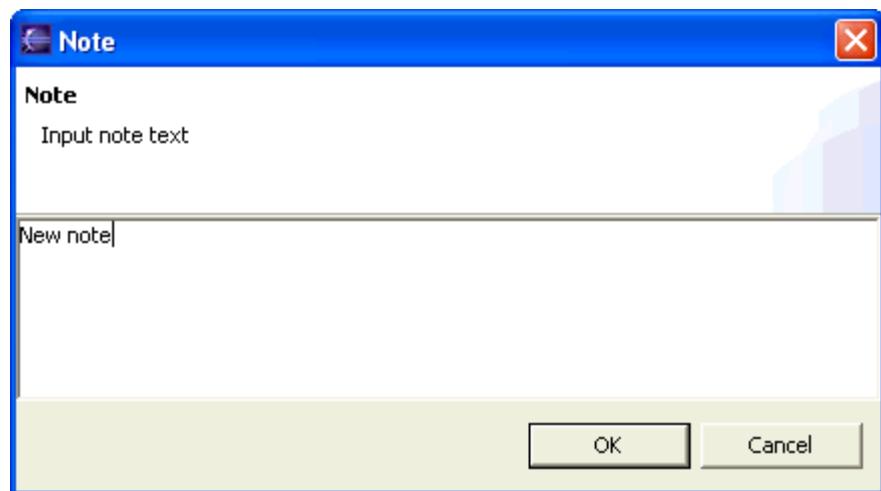
Notes

New note

Select the note icon in the tool bar and click inside the diagram. The note will be created at the mouse position.

A dialog allows you to edit the note.

You can change the content by double clicking on the note in the diagram.



New indication

Select the indication icon in the tool bar. The indication will link the note to any diagram element.



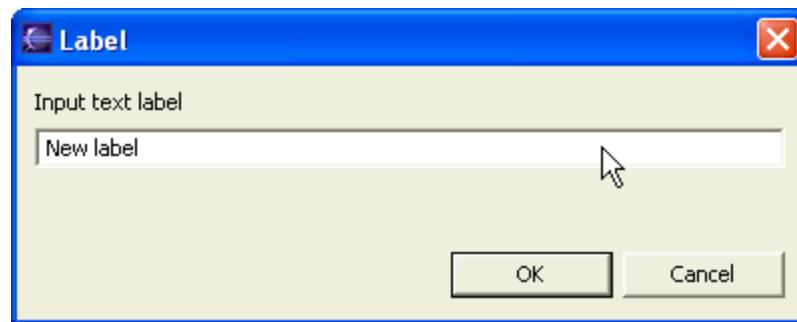
Label

A New label

Select the label icon in the tool bar and click inside the diagram.

A dialog allows you to edit the label.

You can change the content by double clicking on the label in the diagram.



The label will be created at the mouse position.



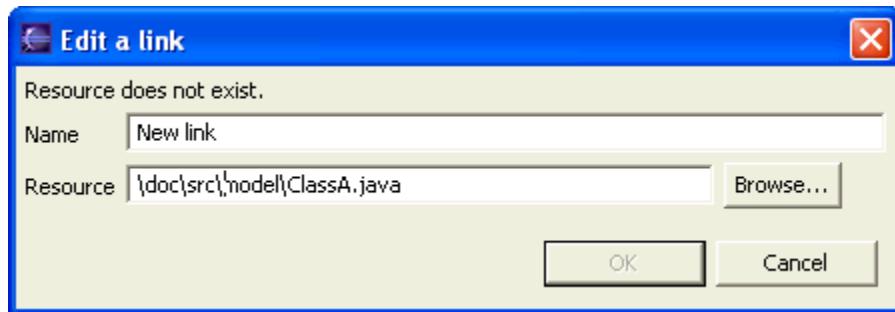
Links

 New link from the tool bar

Select the link icon in the tool bar and click inside the diagram.

A dialog allows you to edit the link.

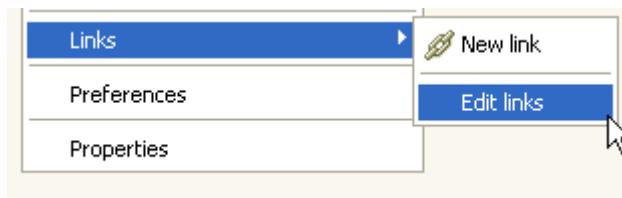
A link can be associated to any element of the workspace.



The link will be created at the mouse position. The displayed icon indicates the kind of resource which has been linked.

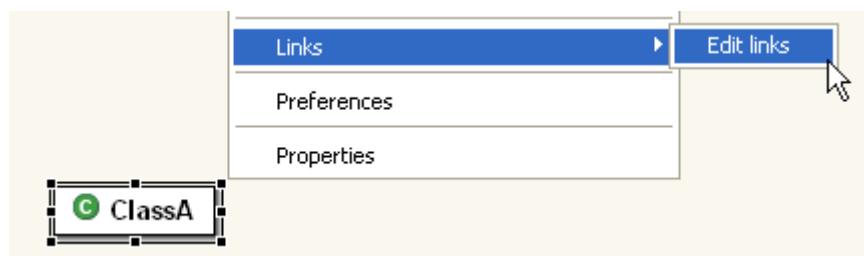


The link can be accessed and edited through the diagram pop-up menu.

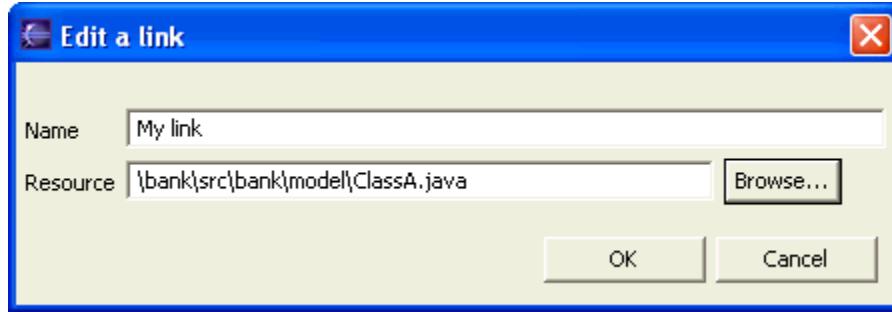


New link from a diagram element

Select any element inside your diagram, open the **popup menu > Links > edit links**



Enter the name of the link in the Name field and select the resource which will be linked to your diagram element.



For example, ClassA has a new link named My link.

You can navigate from ClassA element to another element of the workspace.

Open the **popup menu** > **Links** > **My link**



Key Bindings

Here are the most useful key bindings to manipulate EclipseUML diagrams

Keys	Description
Ctrl+s	Save
Ctrl+a	Select all
Ctrl+z	Undo
Ctrl+y	Redo
Ctrl+p	Print
Ctrl+n	new wizard
Ctrl+F4	Close the diagram editor
Delete	Delete the selected figures with model
.	Enter the resize and moving mode. Repeating the key can switch different operations such as resize to left/right/up/down and moving.
Escape	Exit the resize and moving mode
tab	Change the tool bar selection
Ctrl+space	select or de-select a figure
Arrow	Change the figure selection with auto-scrolling
Shift+arrow	Append the figure selection
Ctrl+arrow	Change the focus
Ctrl+Alt+arrow	Move the selected figure
Ctrl+Shift+D	Launch the "Open Diagram" dialogue to select a diagram in the project and open it

Diagrams

Class Diagrams

The following areas are covered:

- Concept
- Drag'n'Drop
- Presentation Mode
- Navigation
- Real-Time Synchronization
- Refactoring
- Working At Package Level
- View Selector
- Property Concept
- Toolbar
- Class Diagram Example

Concept

Class Diagrams are static and show an overview of a system. A system is composed of classes and the relationships among them.

The Class Diagram model describes the static structure of the symbols in the system. This model allows you to graphically represent symbol diagrams containing classes. Classes are arranged in hierarchies sharing common structure and behavior and are associated with other classes.

UML class diagrams show the classes of the system, their inter-relationships, and the operations and attributes of the classes.

Class diagrams are used to:

- analyze requirements in the form of a conceptual/analysis model
- explore domain concept in the form of a domain model
- depict the detailed design

In a class diagram different elements are available:

1. **New Packages** can be created in the Class Diagram Editor :

-  Using the class diagram toolbar.
-  Using the class diagram editor's popup menu (Right click inside the class diagram editor >insert>package).
-  Using drag and drop from the Package Explorer to the class diagram editor.

The **look and feel** of the package elements can be changed depending on the selected diagram presentation style. Three different diagrams presentation styles can be applied to the Class Diagram Editor :

- Omndo presentation



- UML Presentation



- Eclipse Presentation



2. **New Classes** can be created in the Class Diagram Editor :

-  Using the class diagram toolbar.
-  Using the class diagram editor's popup menu (Right click inside the class diagram editor >new>class).
-  Using the class diagram editor's popup menu (Right click inside the class diagram editor >insert>class).
-  Using drag and drop from the Package Explorer to the class diagram editor.

The **look and feel** of the class elements can be changed depending on the selected diagram presentation style. Three different diagrams presentation styles can be applied to the Class Diagram Editor :

- Omndo presentation



- UML Presentation



- Eclipse Presentation



3. New Interfaces can be created in the Class Diagram Editor :

- Using the class diagram toolbar.
- Interface Using the class diagram editor's popup menu (Right click inside the class diagram editor >new>Interface).
- Interface Using the class diagram editor's popup menu (Right click inside the class diagram editor >insert>class).
- Using drag and drop from the Package Explorer to the class diagram editor.

The **look and feel** of the interface elements can be changed depending on the selected diagram presentation style. Three different diagrams presentation styles can be applied to the Class Diagram Editor :

- Omndo presentation



- UML Presentation



- Eclipse Presentation

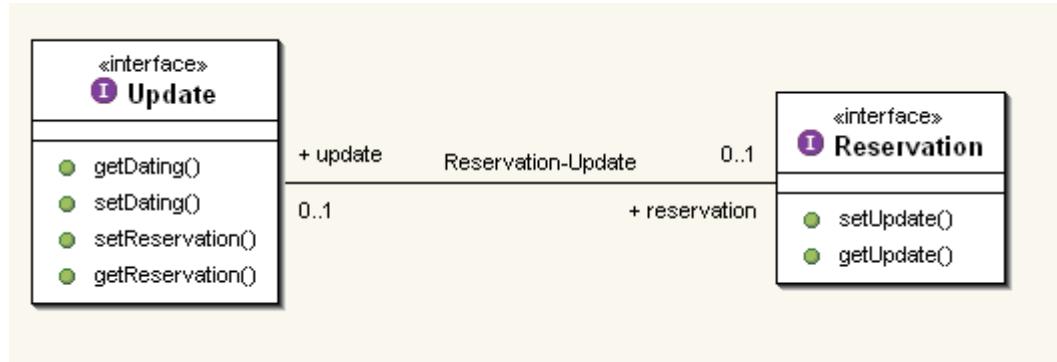


4. **New Associations** between classes and interfaces can be created in the Class Diagram Editor :
 Using the class diagram toolbar.

The creation of a association between classes and interfaces has four steps using the mouse:

1. Select the association icon in the toolbar.
2. Select a class or an interface (the color of the element becomes blue, that means it is selected), this is the first association end.
3. Hold down the mouse button and drag the link from the first element to the second, this is the second association end.
4. Release the mouse button.

The properties of the new association are defined through the [preferences](#).



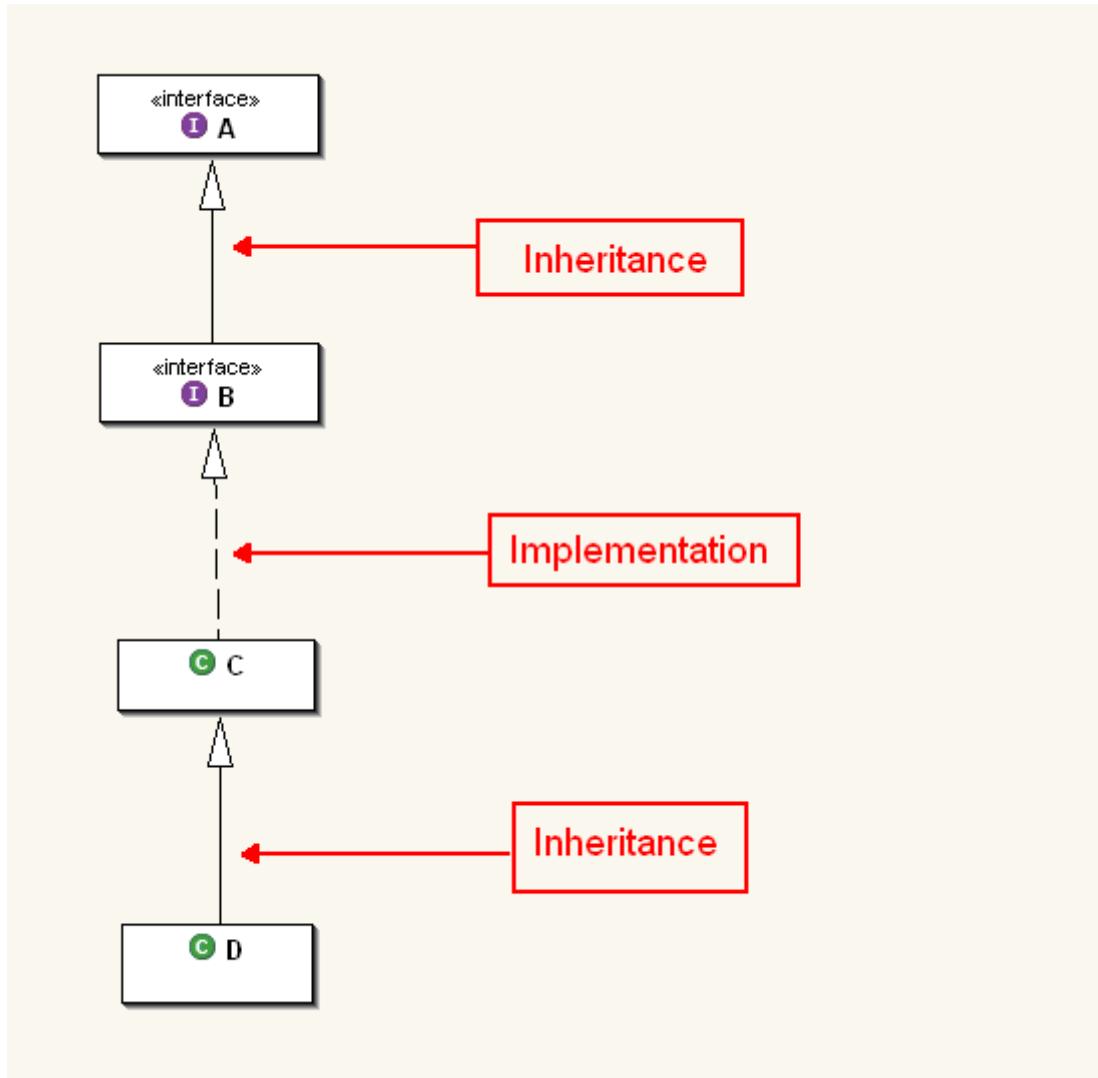
5. **New Generalizations** between classes and interfaces can be created in the Class Diagram Editor :
 Using the class diagram toolbar.

The creation of a generalization between classes and interfaces has four steps using the mouse:

1. Select the generalization icon in the toolbar.
2. Select a class or an interface (the color of the element becomes blue, that means it is selected).
3. Hold down the mouse button and drag the link from the first element to the second.
4. Release the mouse button.

Depending on the nature of the selected elements, the created generalization is an inheritance or an implementation.

The properties of the new generalization are defined through the [preferences](#).



6. New Dependencies between any element can be created in the Class Diagram Editor :

- ↗ Using the class diagram toolbar.

The creation of an dependency between two elements has four steps using the mouse:

1. Select the dependency icon in the toolbar.
2. Select an element (the color of the element becomes blue, that means it is selected).
3. Hold down the mouse button and drag the link from the first element to the second.
4. Release the mouse button.

The properties of the new dependency are defined through the [preferences](#).

Drag and Drop

In a class diagram, you can drag and drop the following elements from the Package Explorer to the Class Diagram Editor:

- Package
- Class
- Interface

Presentation

When a UML Class Diagram is created, the diagram presentation mode can be selected. The diagram presentation mode allows you to focus on what it is important to show inside the new Class Diagram.

Three kinds of elements are provided :

1. Association
2. Inheritance
3. Dependency

We recommend selecting just one element at a time in order to create a better presentation. All elements could be selected at once if needed, this is mostly useful for a simple class diagram presentation.

Having too much information in the diagram is not recommended.

Navigation

Different navigations are possible using EclipseUML.

- [1. Navigation from a Class](#)
 - [1.1 To a Class-Centric Diagram](#)
 - [1.2 To a State Diagram](#)
 - [1.3 To a Sequence or Collaboration Diagram](#)
- [2. Navigation from a package](#)
 - [2.1 To a Package-Centric diagram](#)
 - [2.2 To a Class Diagram](#)

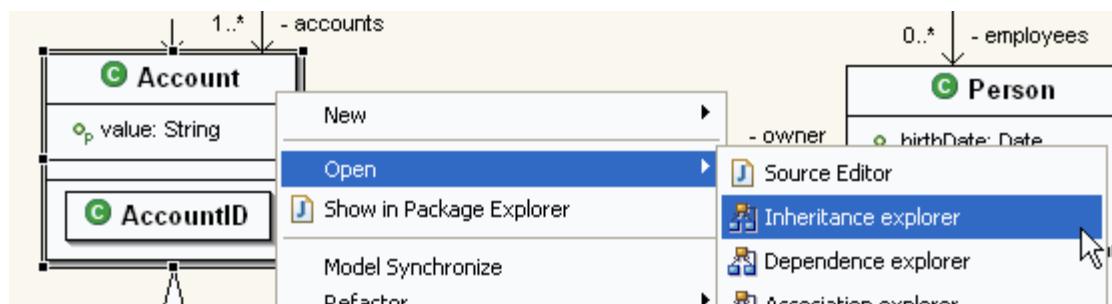
1. Navigation from a Class

1.1 To a Class-Centric Diagram

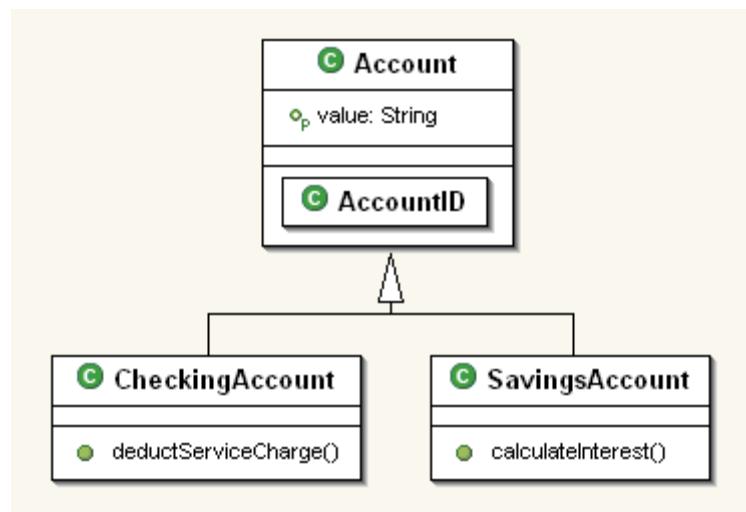
Navigation from a class to a class-centric diagram is available with 4 options :

- **Inheritance default explorer**

Select a class or an interface, open the popup menu and select **Open>Inheritance explorer**

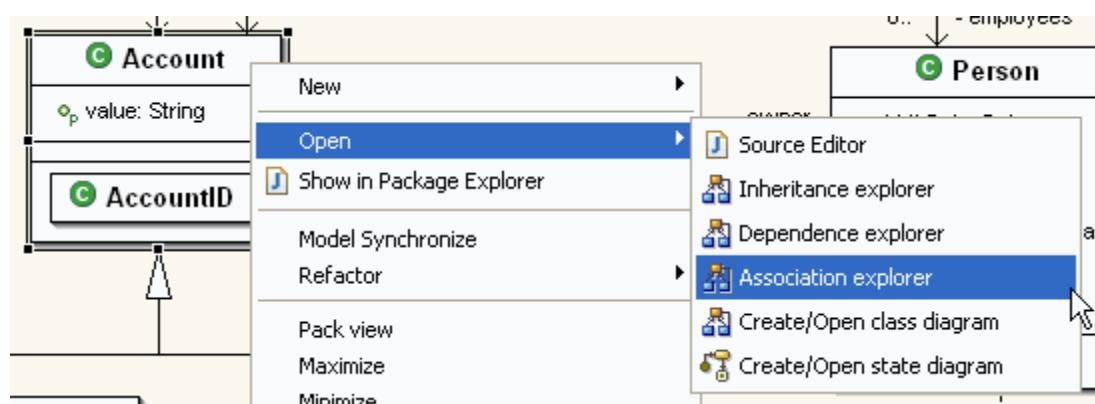


The default Inheritance explorer diagram display the immediate (level 1) inheritance environment of the selected class or interface.

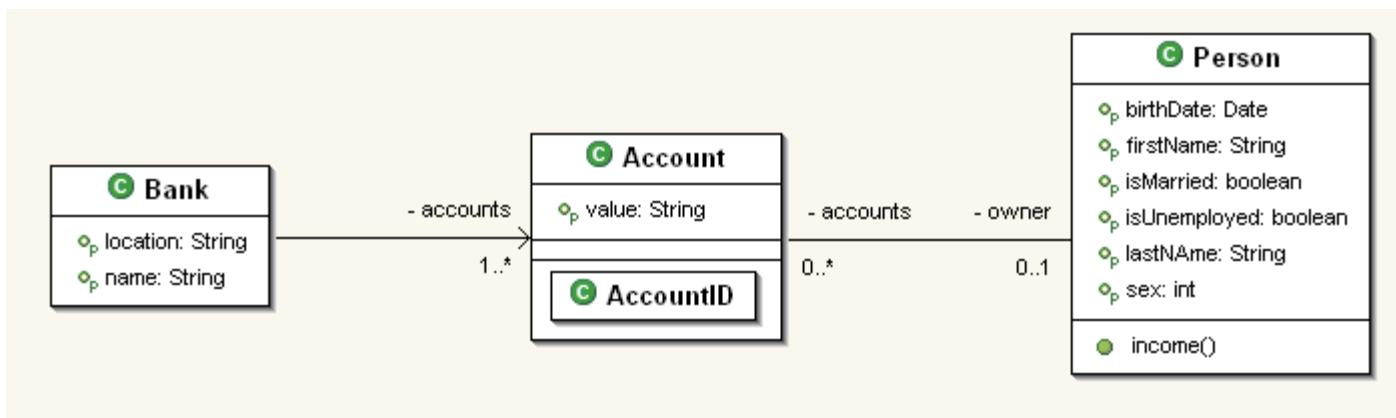


- **Association default explorer**

Select a class or an interface, open the popup menu and select **Open>Association explorer**

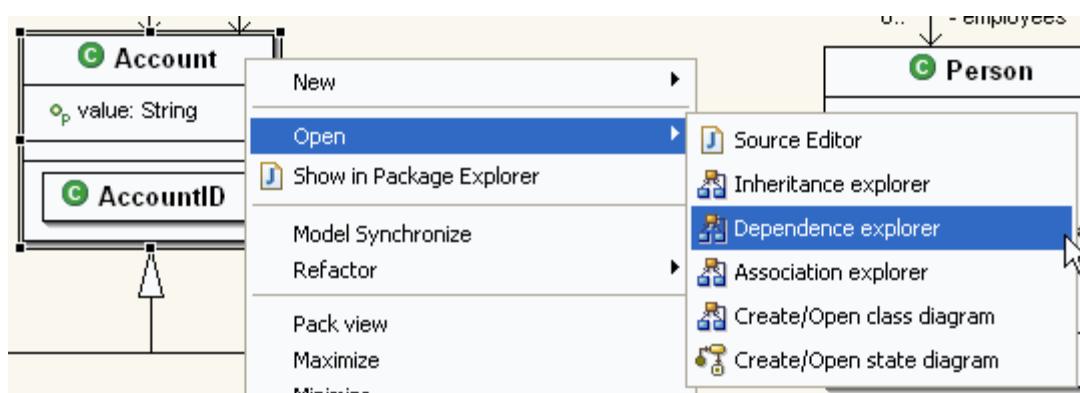


The default Association explorer diagram display the immediate (level 1) associations environment of the selected class or interface.

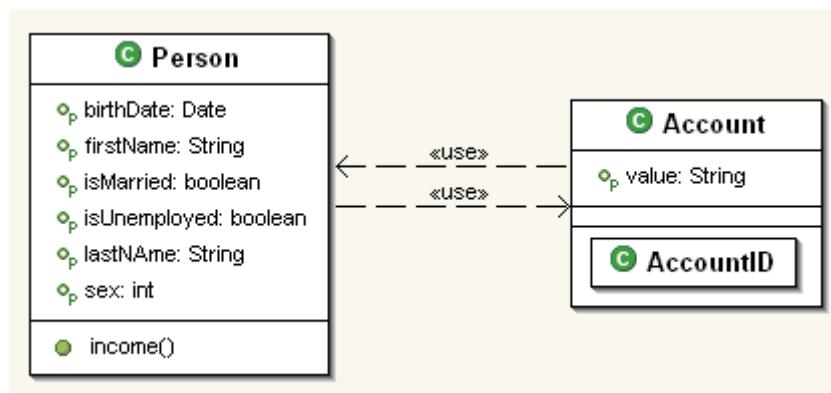


- **Dependency default explorer**

Select a class or an interface, open the popup menu and select **Open>Dependency explorer**



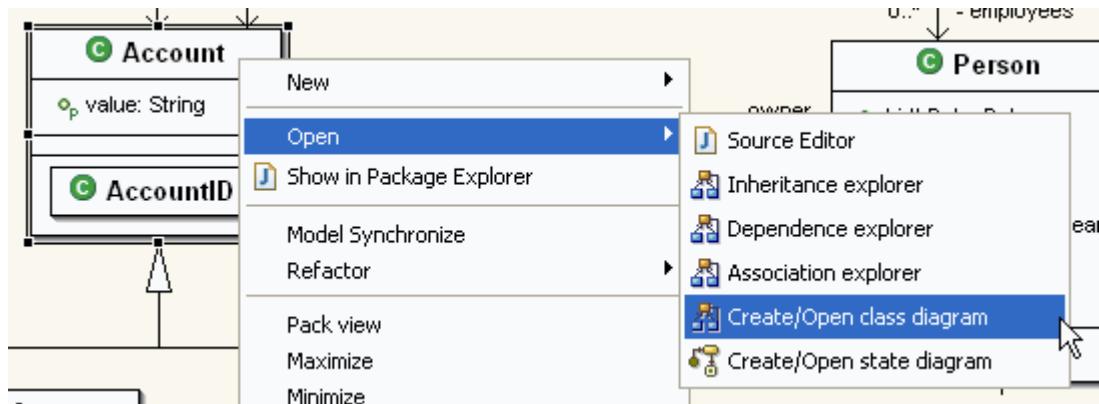
The default Dependency explorer diagram display the immediate (level 1) dependencies environment of the selected class or interface.



- **Customizable Class-centric Diagram**

Open new class diagram and navigate within a Scope (package, project, all) and from -1 (infinite) to n levels

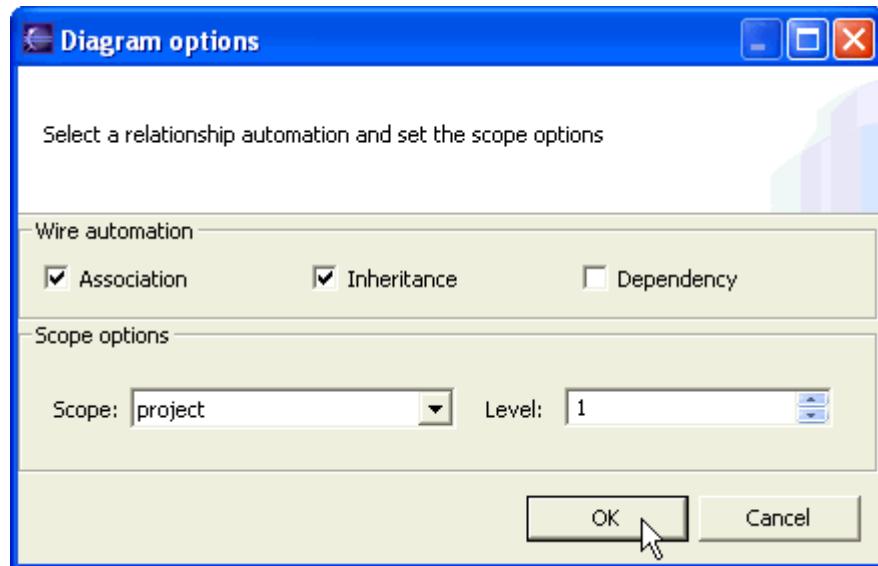
Select a class or an interface, open the popup menu and select **Open>Create/open class diagram**



Associations, interfaces and/or dependencies can be displayed in the diagram through the **Wire automation** option.

The diagram scope can be selected through the **Scope** option. Three options are available :

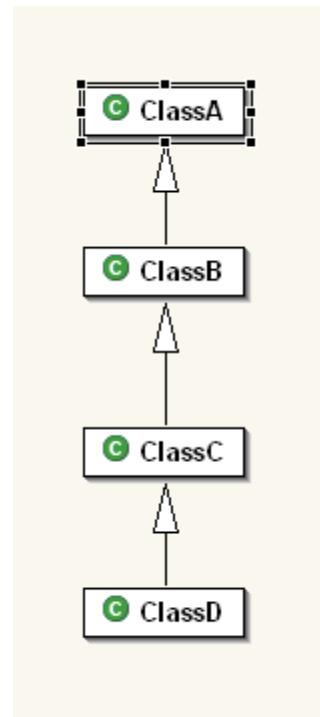
- package
- project
- all



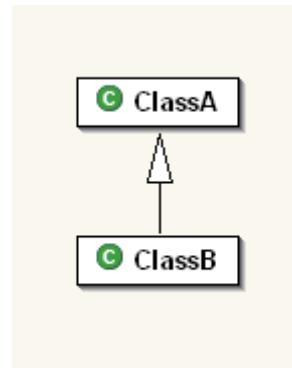
The diagram level can be selected through the **Level** option.

Here are some examples :

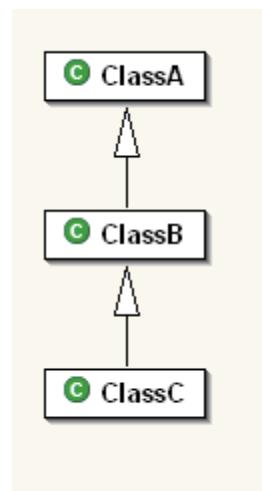
Level -1 will analyse at infinite level and will then show all classes (Class A....Class D)



- 0 will analyse at the class level and will show Class A
- 1 will analyse at one level and will show ClassA and ClassB



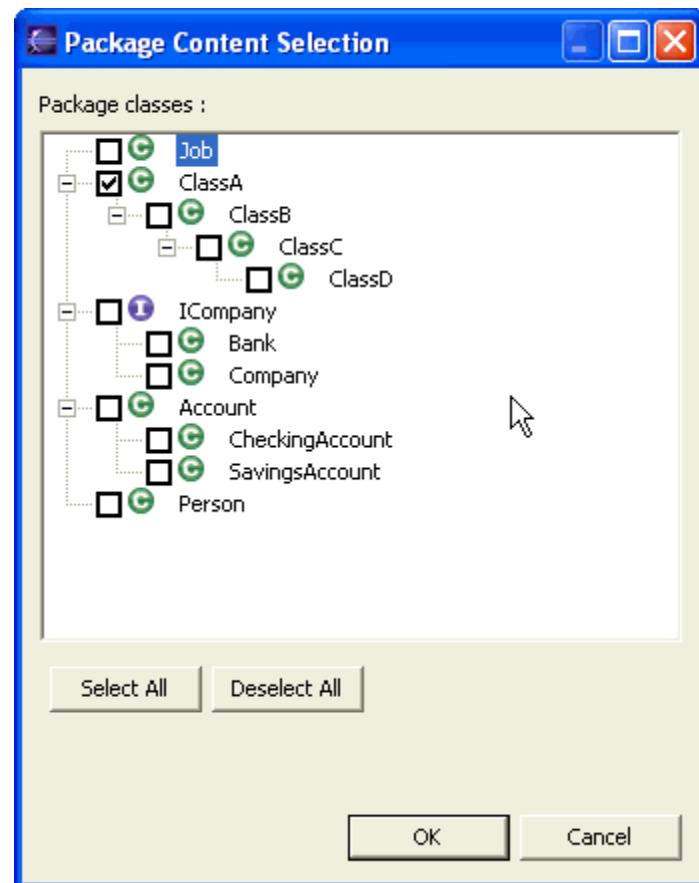
- 2 will analyse at depth 2



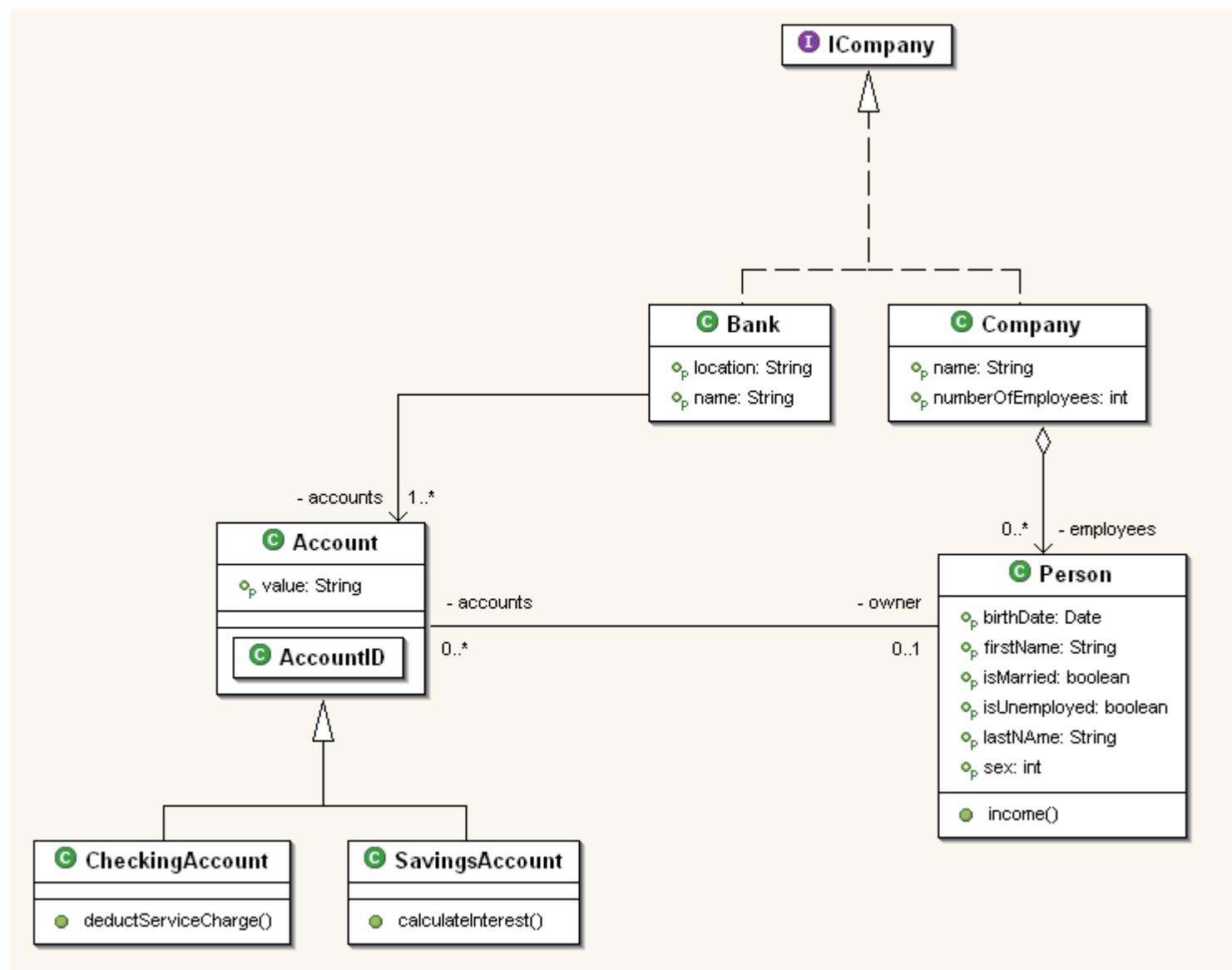
If you select an element, n level from the selected element will be displayed.

When working on large project, it is not recommended to create a diagram with scope "all" and level - 1 which would be a full project diagram (the process could be heavy).

After selecting the option, a class selection dialog appears :



The expected diagram id created :



1.2 To a State Diagram

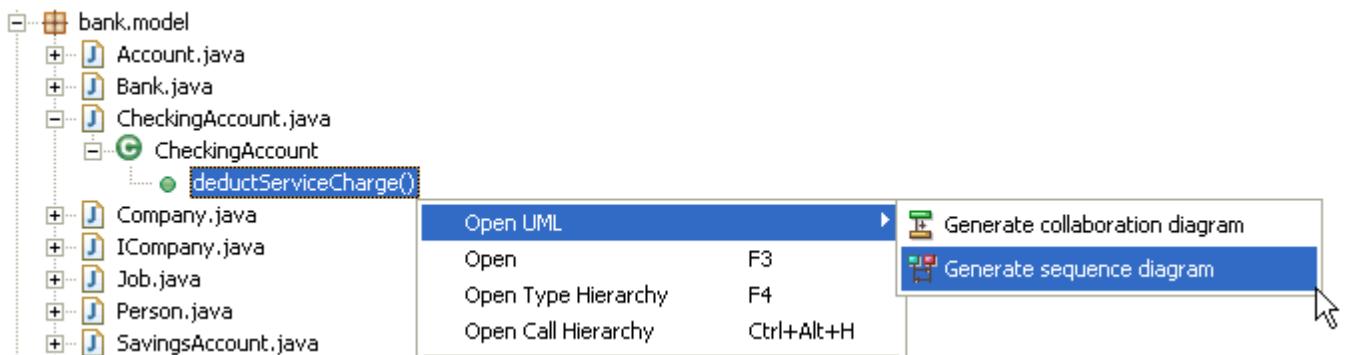
You can navigate from a class or an interface to a State diagram.

Select a class or interface, open the popup menu : **State diagram editor**

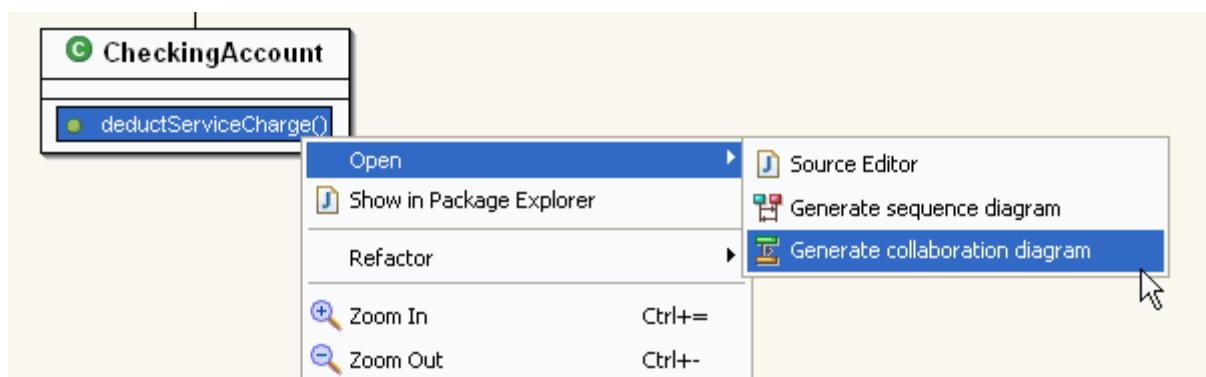


1.3 To a Sequence or Collaboration Diagram

You can navigate from a method to a Sequence/collaboration diagram.
Select a method, open the popup menu : **Generate sequence diagram**



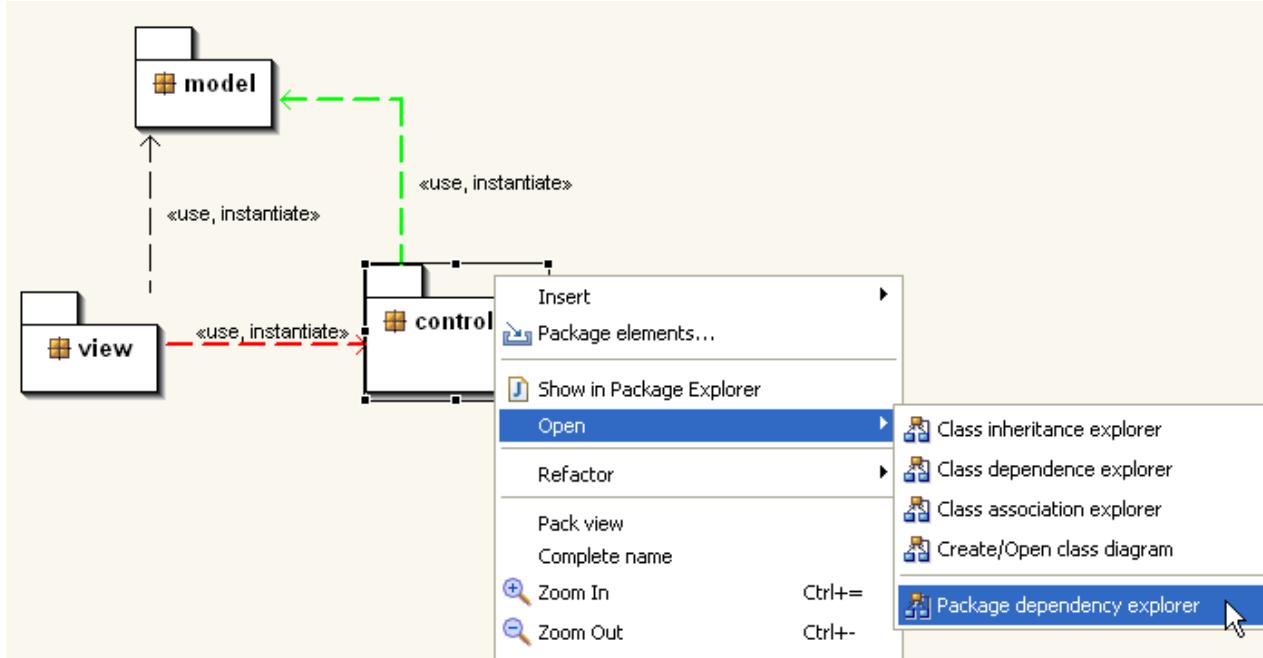
Select a method, open the popup menu : **Generate collaboration diagram**.



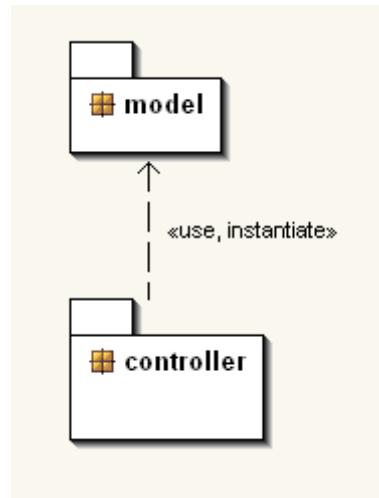
2. Navigation from a Package

2.1 To a Package-Centric Diagram

Select a package, open the popup menu : **Open>Package dependency explorer**



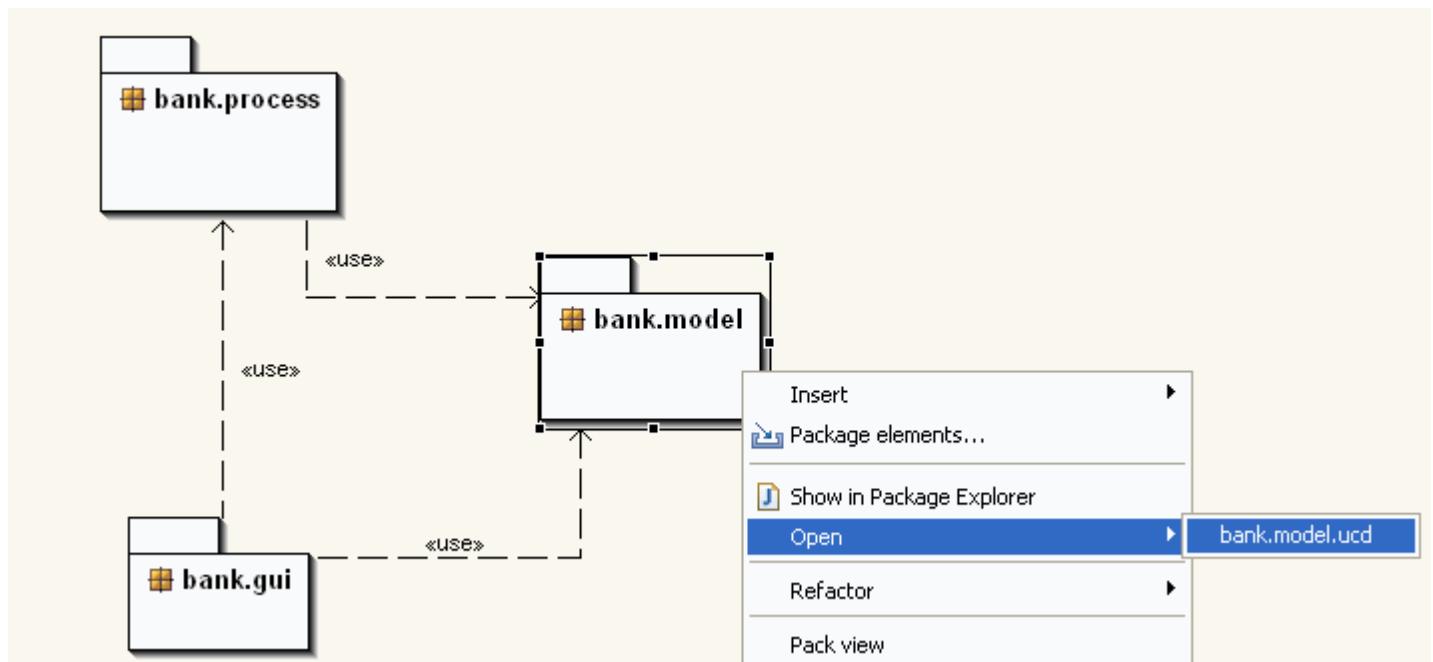
The default package dependency explorer focus on the selected package and only display its immediate outgoing dependencies environnement.



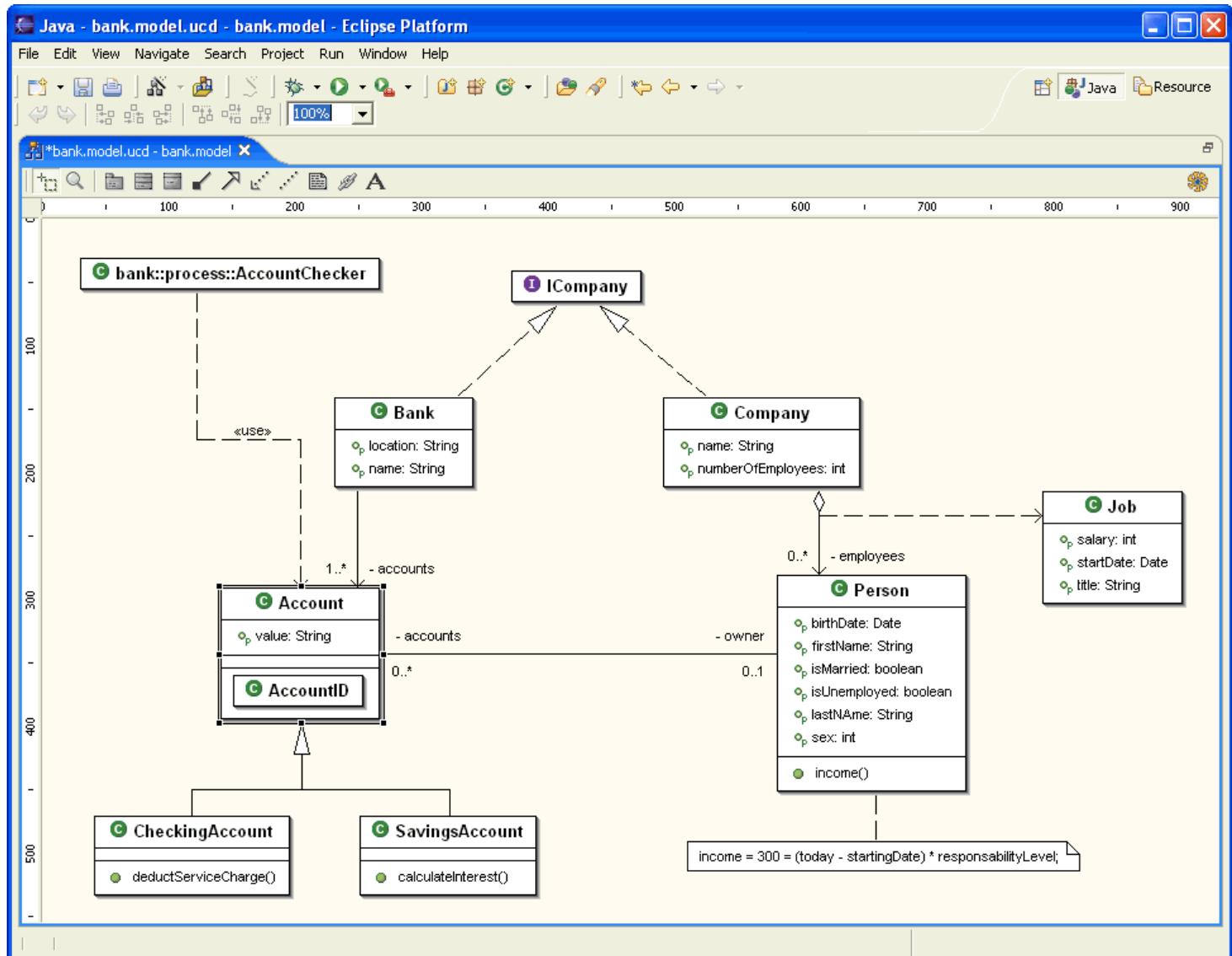
2.2 To a Class Diagram

The navigation is the opportunity to browse several UML model. You can access this functionality through the contextual menu on a package.

- by clicking on *Open > myModel* where *myModel* is an existing model in the package.
- **double click on the selected package** will zoom inside and show classes.

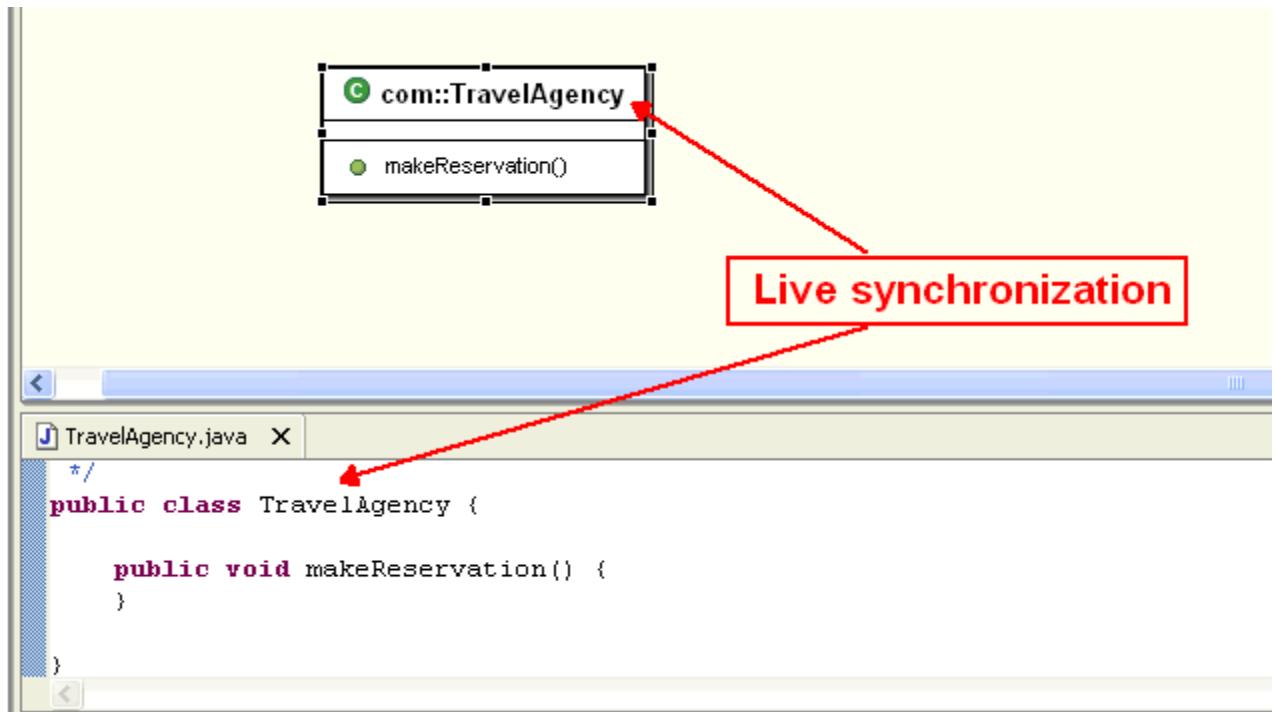


The selected diagram appears in a new tab.



Real-Time Synchronization

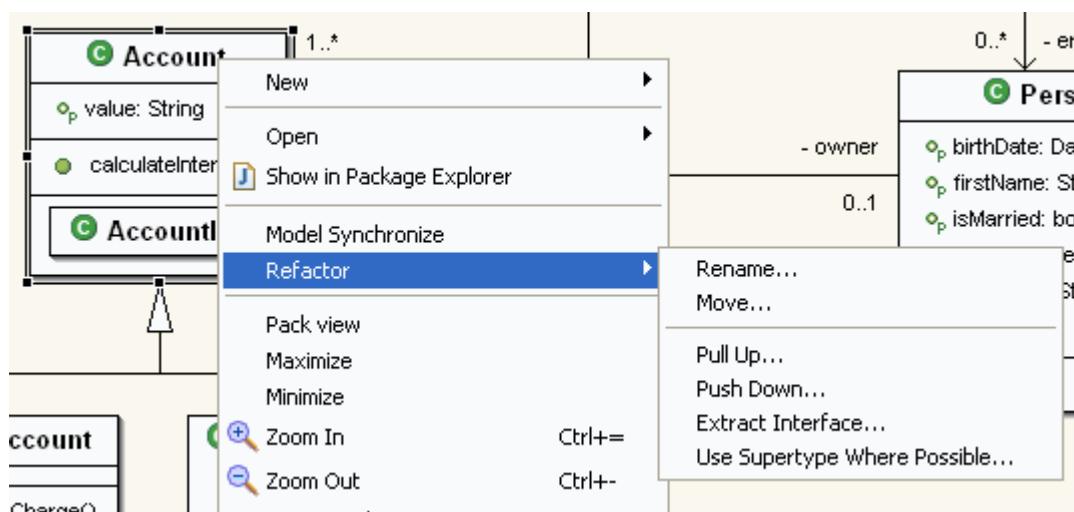
The *EclipseUML plugin* is fully integrated into the Eclipse Framework. EclipseUML is a seamless integrated component in your IDE. You can modify your code either from the Java editor or through the UML view, both are always totally and immediately synchronized.



Refactoring

All Eclipse refactoring functions are available inside the Class Diagram Editor.

To access these functions, select an element in the class diagram editor, right click to open the popup menu and deploy the refactor submenu.

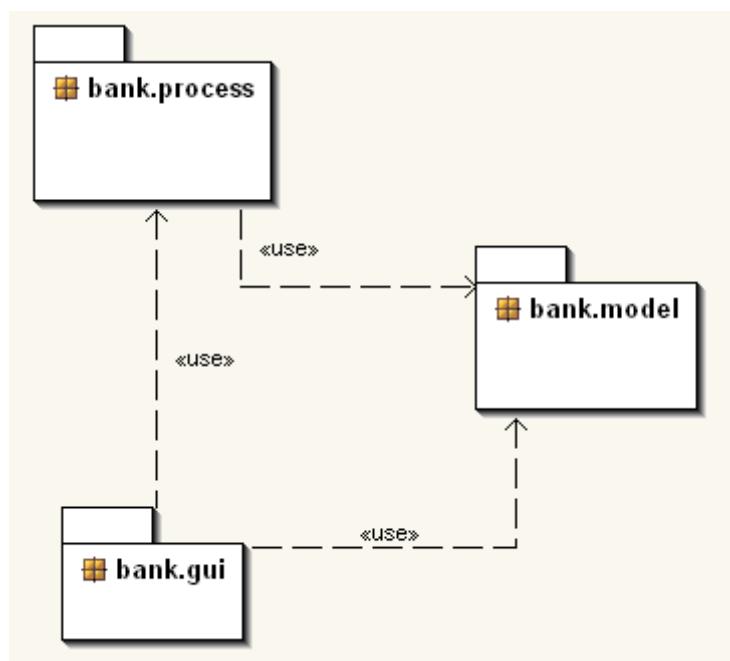


EclipseUML Class Diagram and Java code are immediately synchronized as soon as the refactor function is activated.

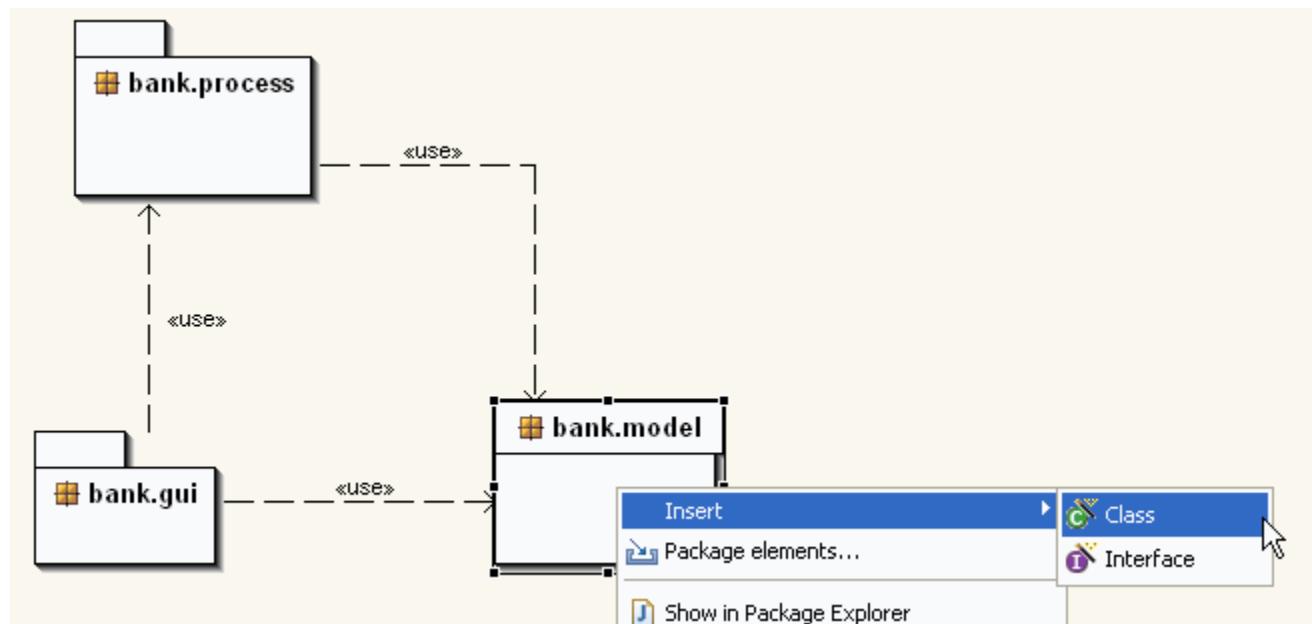
Working at Package Level

Any class diagram created in a package fragment root is a specific diagram dedicated to show the packages and their relationships. The other class diagrams (created in packages) can also work with packages but they must be explicitly inserted in the diagram.

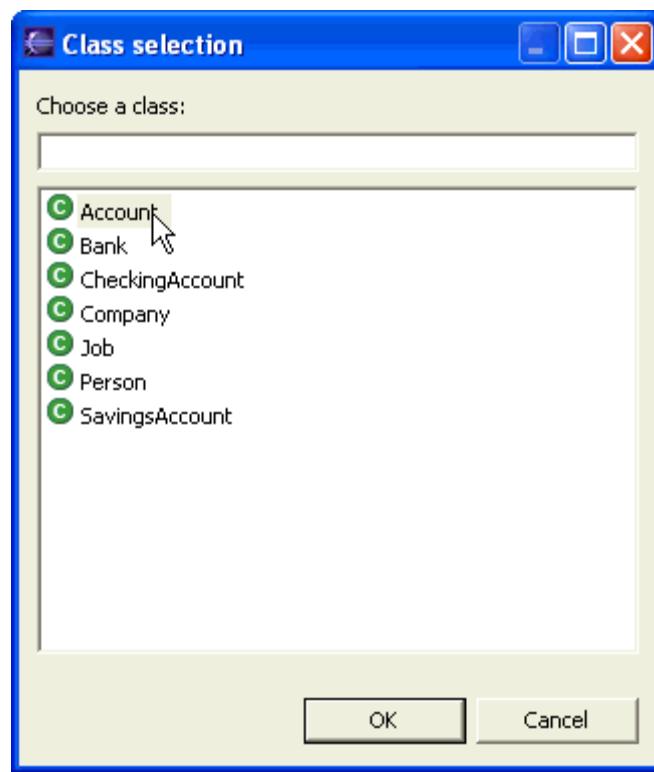
The following screenshot shows an example. Three packages are represented with their dependencies : `model`, `process` and `gui`. Here the package `gui` depends on both packages `model` and `process`, while the `process` package depends on the `model` package.



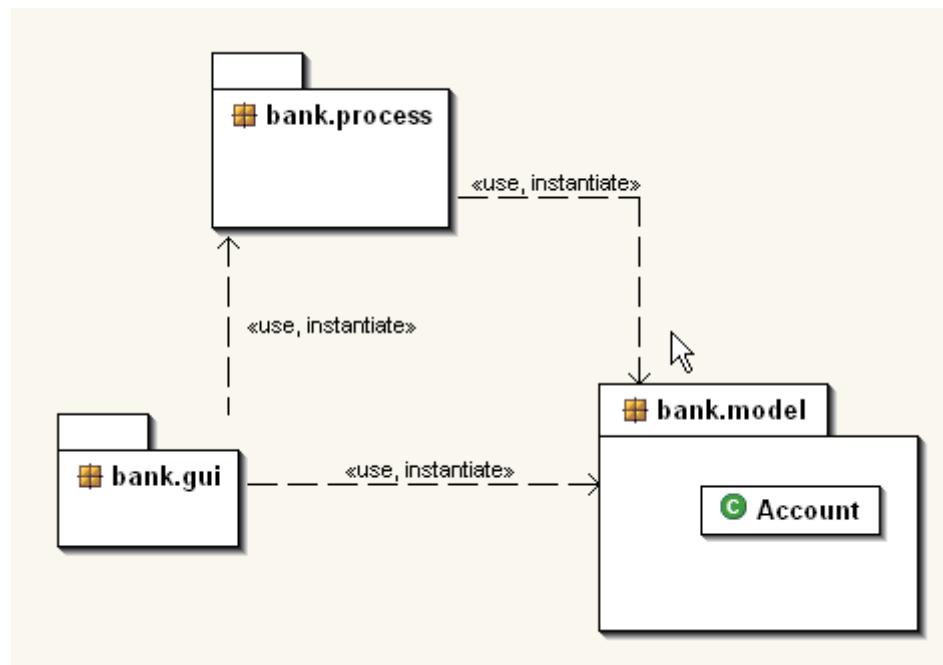
Classes (and Interfaces) can be inserted in the packages by using the contextual menu item *Insert > Class* (or *Insert > Interface*) on the package.



A dialog box proposes to choose one the packages classes (or interfaces).



Once chosen, the class (or interface) appears in the package view.



The layout can be applied to either the whole diagram or a single package with its inserted element.

View Selector

Stereotypes, attributes, methods, inner classes and inner interfaces can be shown or hidden in the Class Diagram packages (only for stereotypes), classes and interfaces. This is done through a window called *View selector*. You can access it from the package, class or interface context menu (right click on the representation in the UML class diagram, then *View selector*).

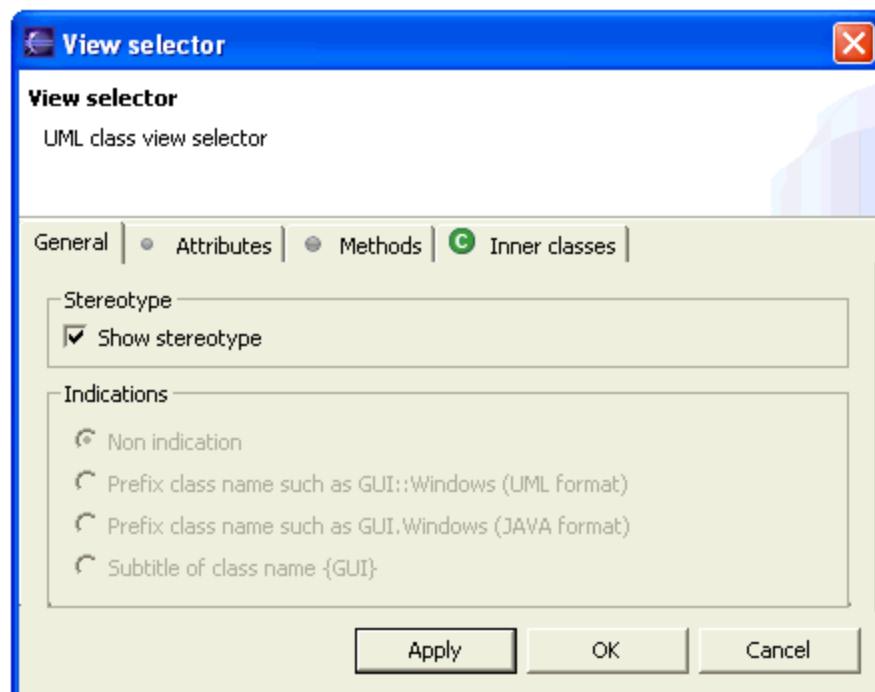
Four options are available:

1. [General](#)
2. [Attributes](#)
3. [Methods](#)
4. [Inner Classes](#)

1.General

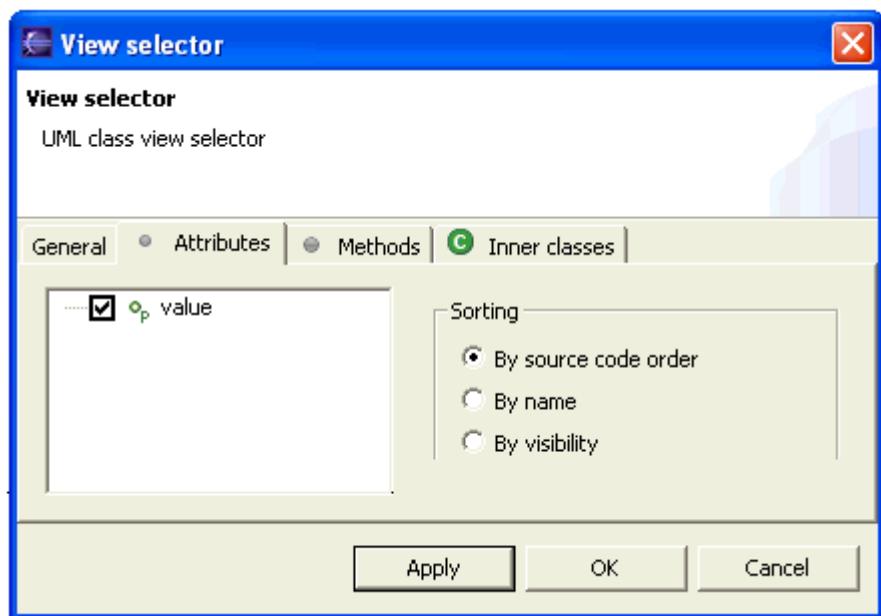
On classes and interfaces, you can configure :

- **Show or not show the stereotype.**
- **Indications :** Allows you to show the class package name in the element. If the class package is the current diagram package, there is no indication.



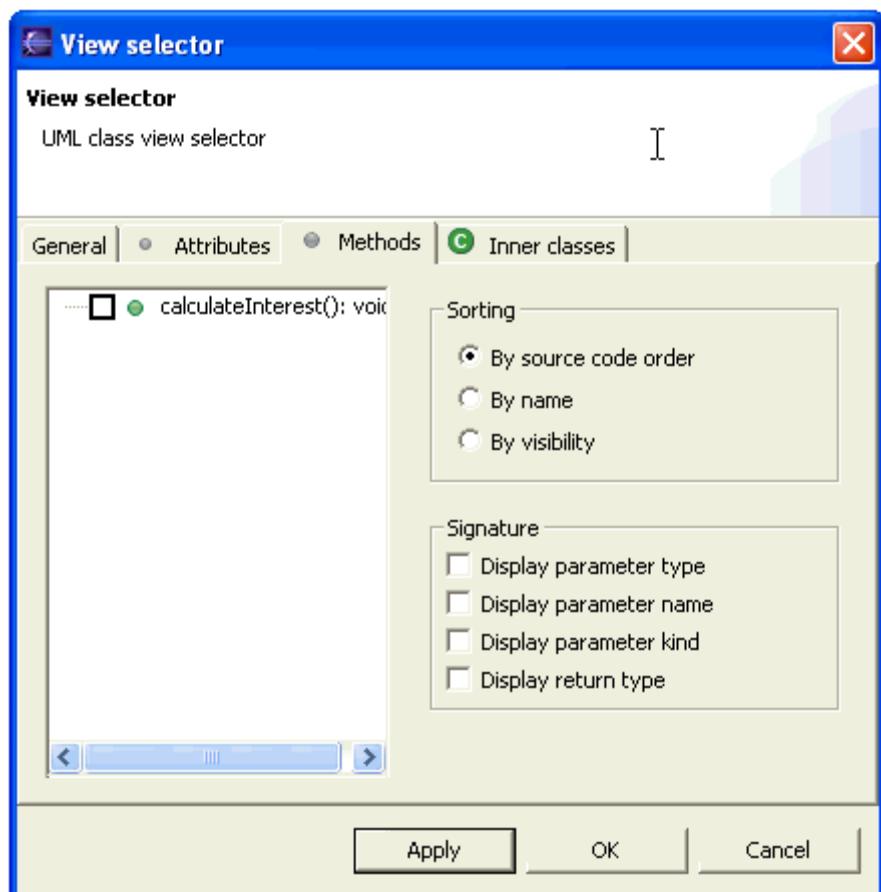
2. Attributes

- Attributes views in the class diagram are set according to the [preferences](#) values, but you can override the behaviour for this class (hide or show attributes) either by (de)selecting directly the attribute, or through the attributes visibility.



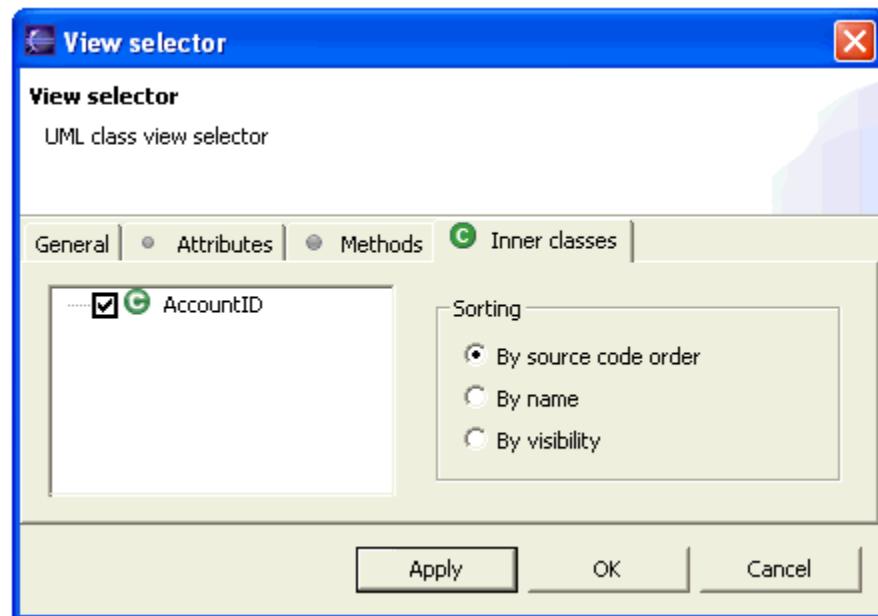
3. Methods

- Methods views in the class diagram are set according to the [preferences](#) values, but you can override the behaviour for this class (hide or show methods) either by (de)selecting directly the method, or through the methods visibility.



4. Inner Classes

- **Inner classes** views in the class diagram are set according to the [preferences](#) values, but you can override the behaviour for this class (hide or show methods) either by (de)selecting directly the method, or through the methods visibility.



Property Concept

When working with a class diagram, it is possible to activate or not the Property concept.

The Property concept is an abstraction of the common characteristics of an attribute and its accessor.

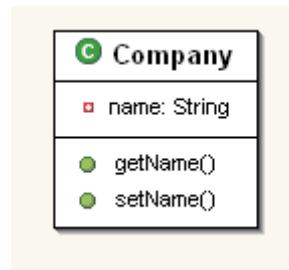
This concept occurs in two contexts :

- In an interface, a property is an abstraction of abstract accessors.
- In a class, a property is either an abstraction of an attribute and its accessors, or an abstraction of abstract accessors.

When activating the property concept, attributes and accessors are represented as a single member of a classifier (class or interface) : a property.



When the property concept is not activated, attributes and accessors are displayed as separated members of the classifier.



Toolbar items



New package

Create a new package in the diagram editor



New Class

Create a new class



New interface

Create a new interface



Association

Create a new association between two elements



Generalization

Create generalization between two elements



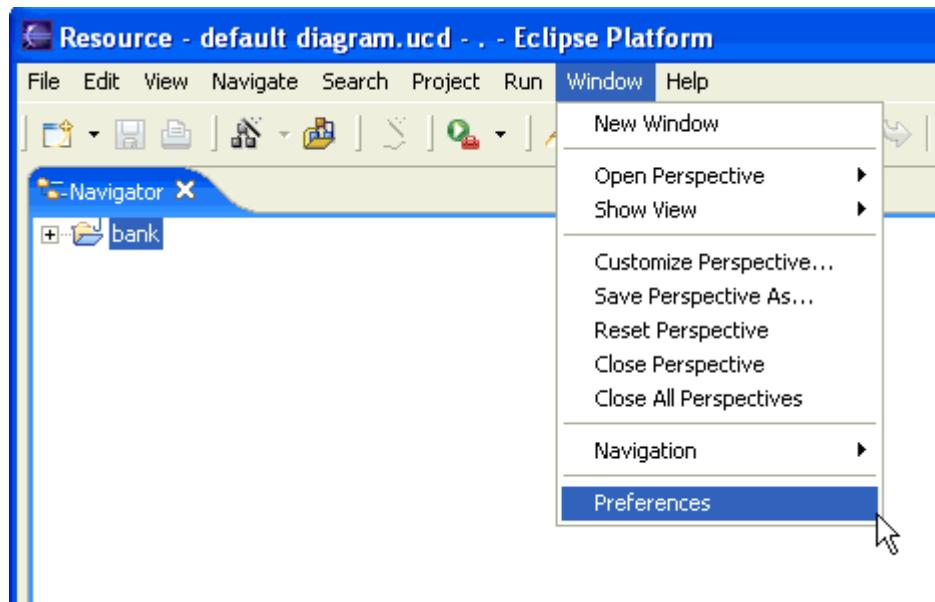
Dependency

Create dependency between two elements

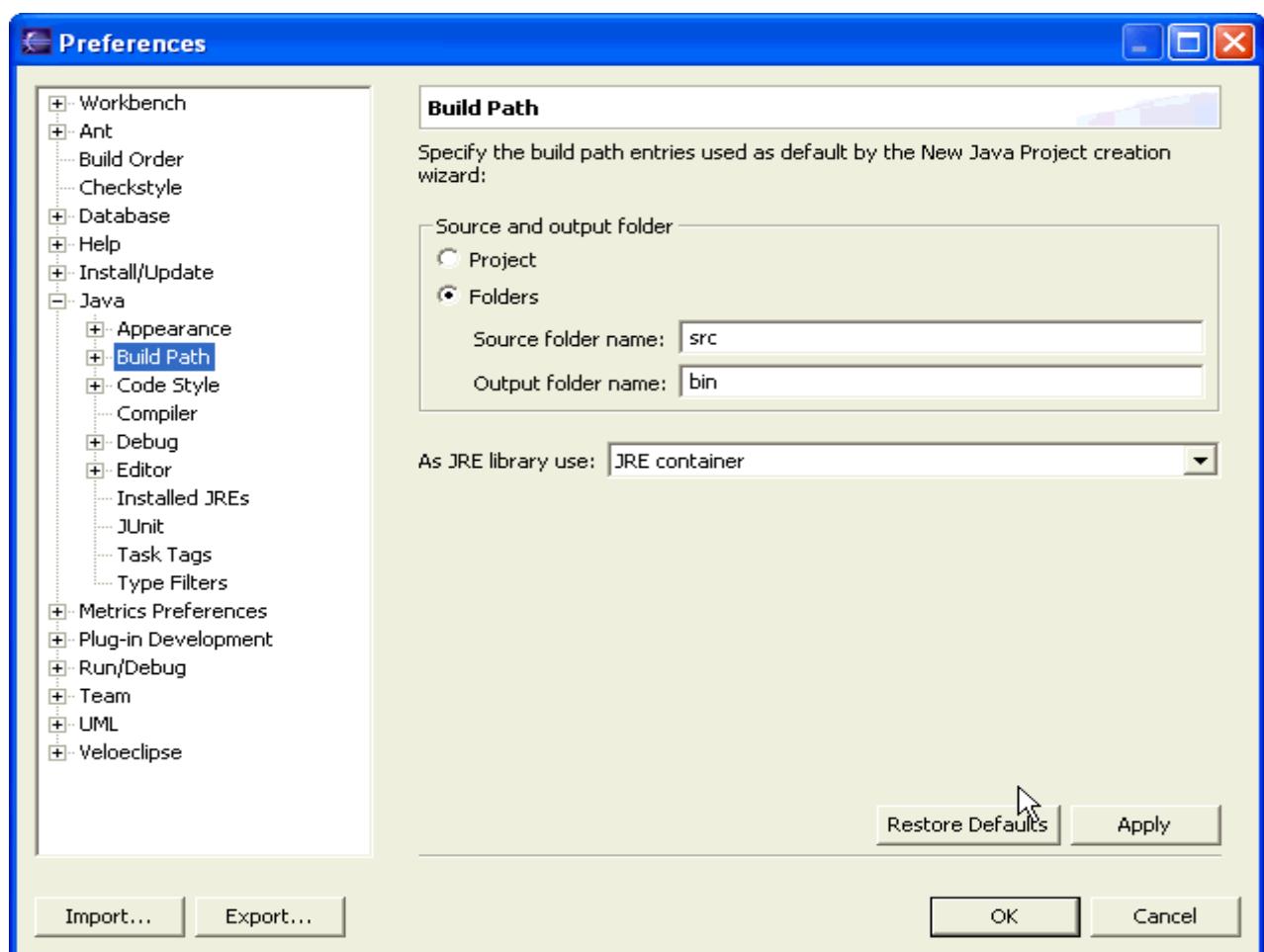
Class Diagram Example

The following example shows a class diagram creation using Eclipse 2.1.

We are going to add general Eclipse Preferences, which will be used in our example.
Click on **Window>Preferences**.



We decided to create a src and a bin output for this project.
Select **Java>Build Path** and click on Folders checkbox.

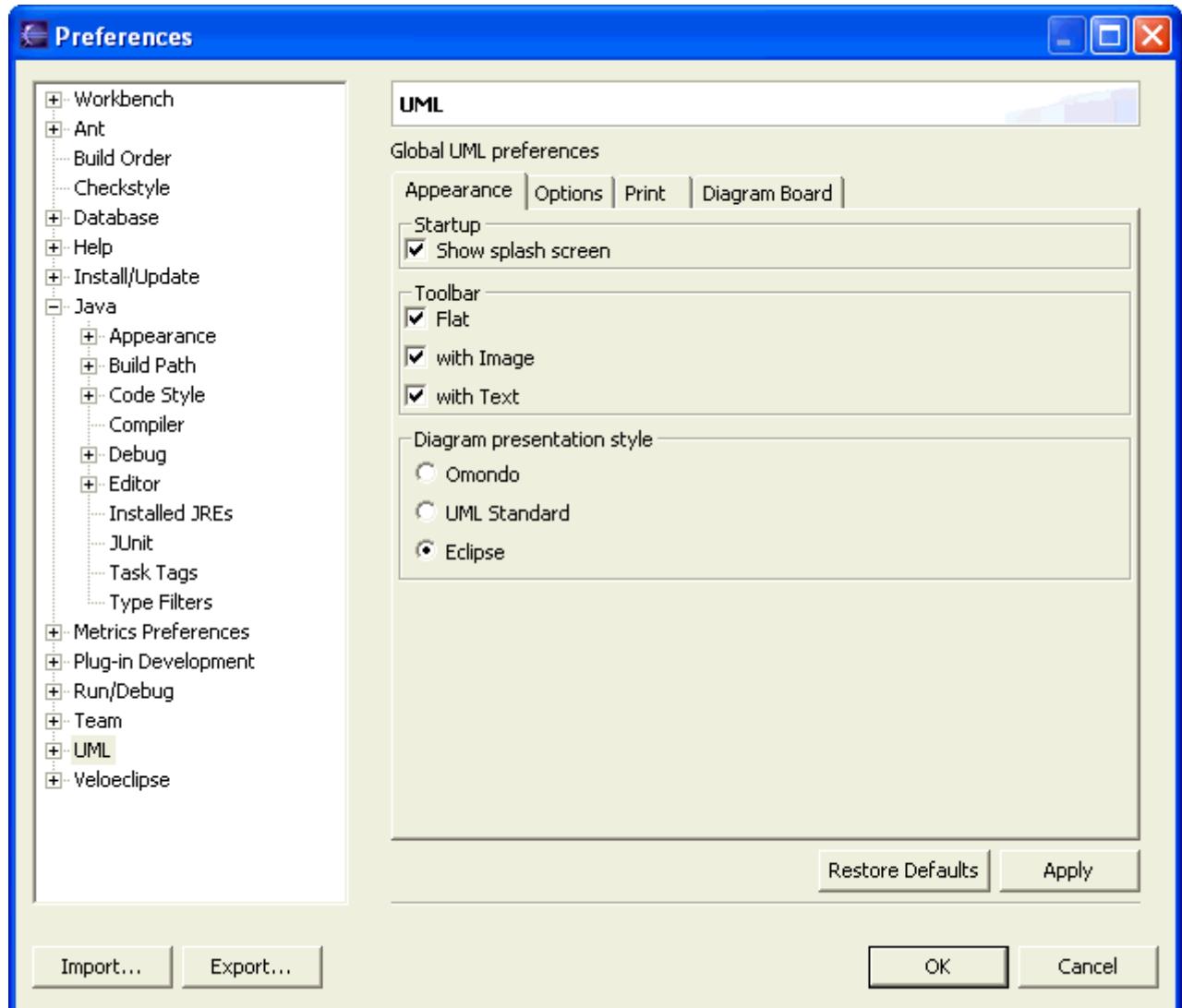


We are going to set up general UML preferences.
 Click on UML, wait for the launch of EclipseUML.
 Select the following options:
 Toolbar:

- Flat
- with image
- with text

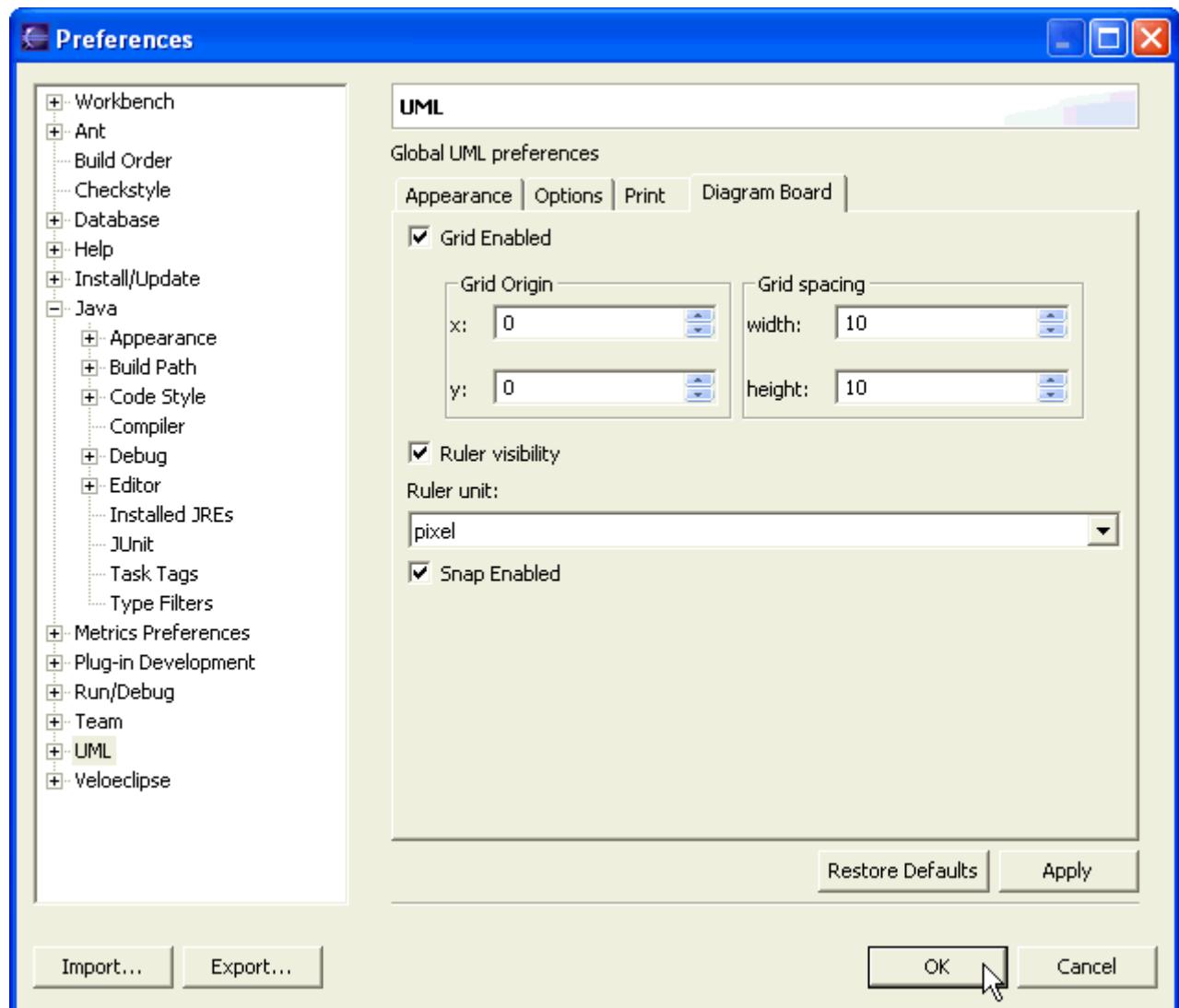
Diagram Presentation:

- Eclipse



Select Diagram Board menu to activate the following options:

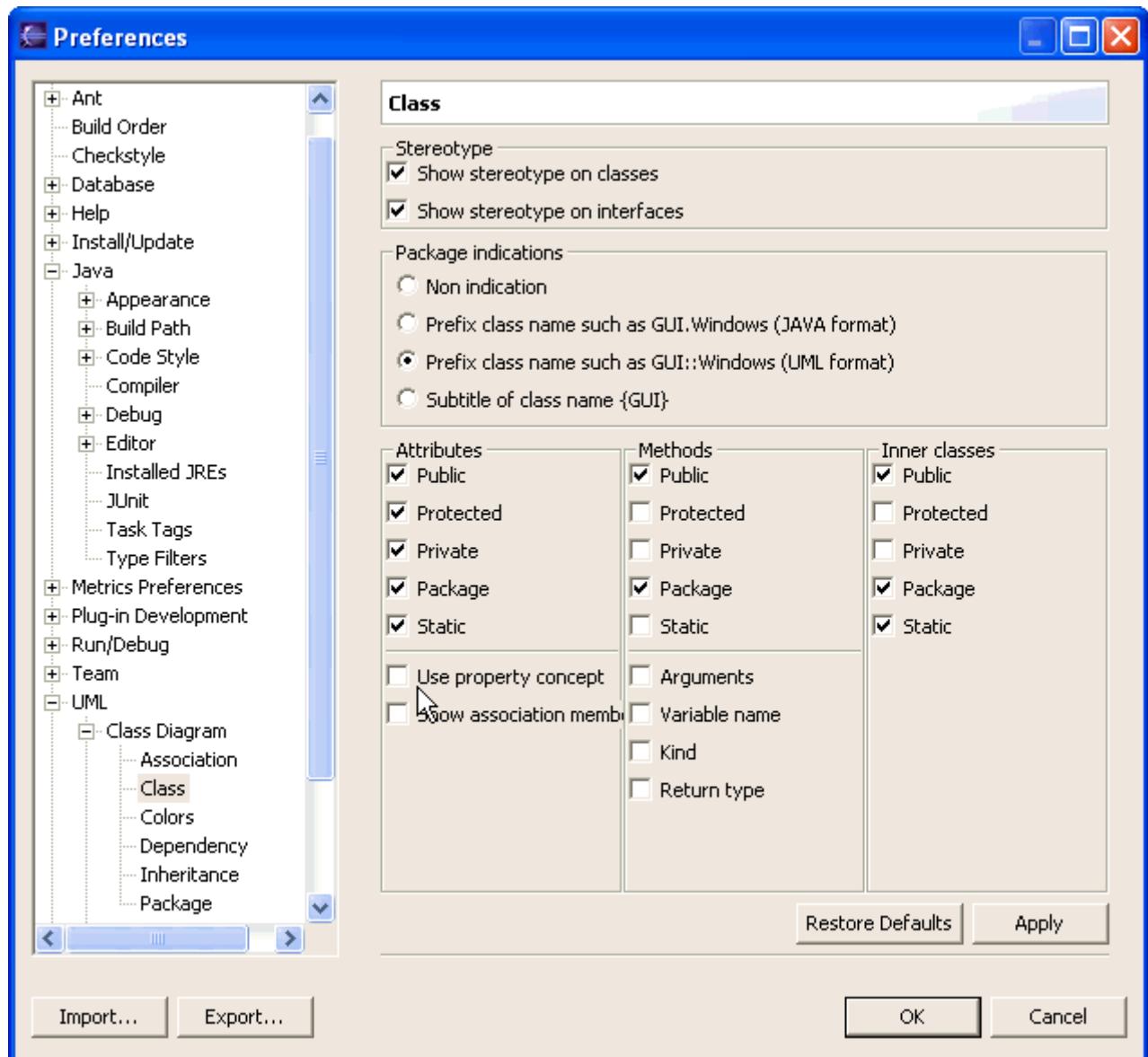
- Grid Enabled
- Ruler visibility > pixel
- Snap Enabled



We want to show Public, Protected, Private, Static and Package Attributes.

We also want to show Public and Package Methods.

We unselect the Use property concept checkbox.

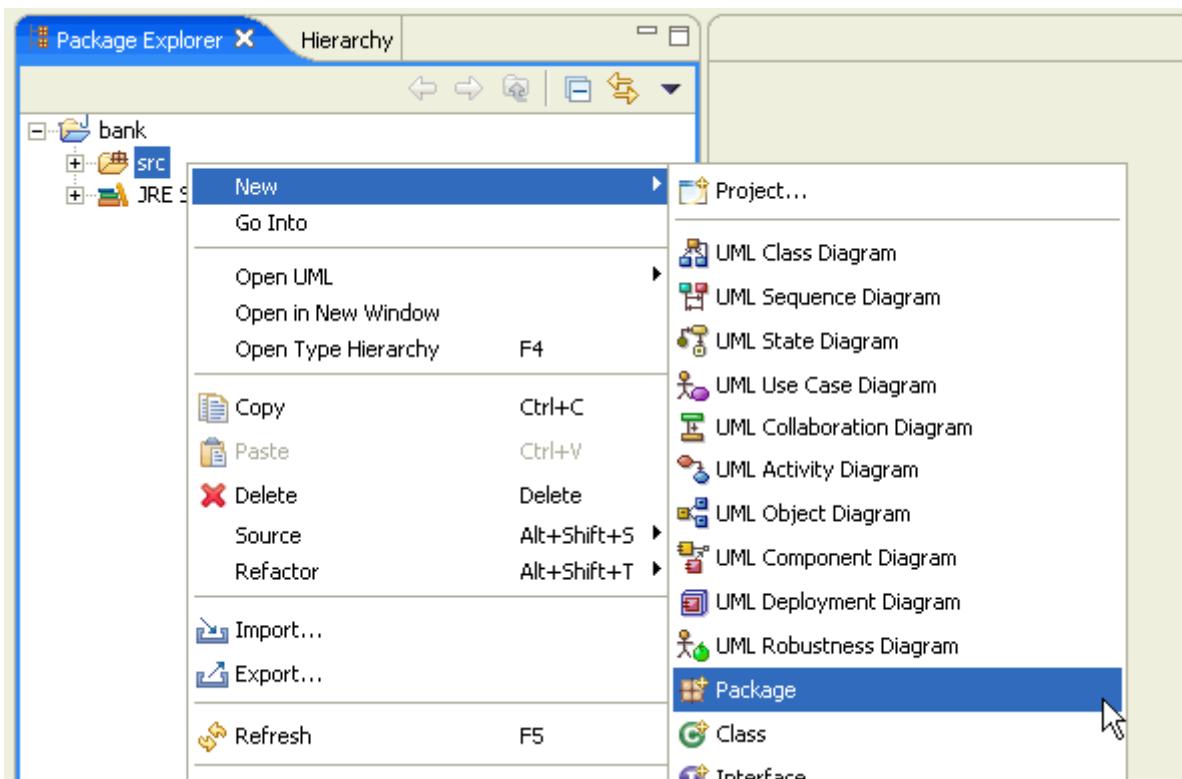


We would like to use a Java Perspective.

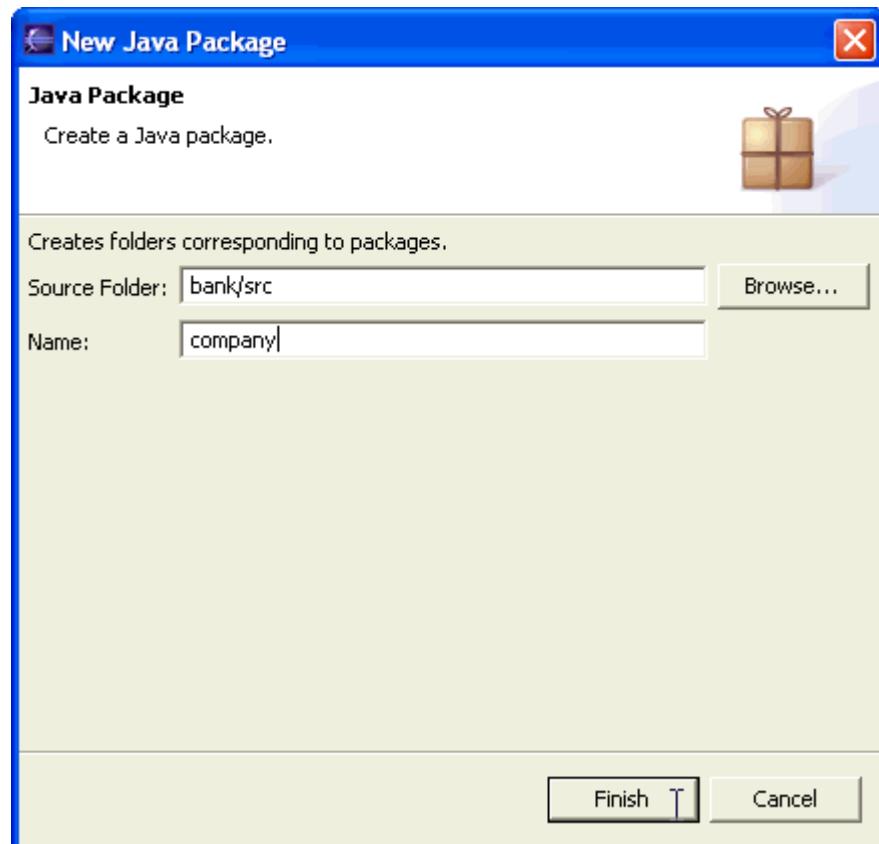
Click on the top icon to **select Java perspective**.



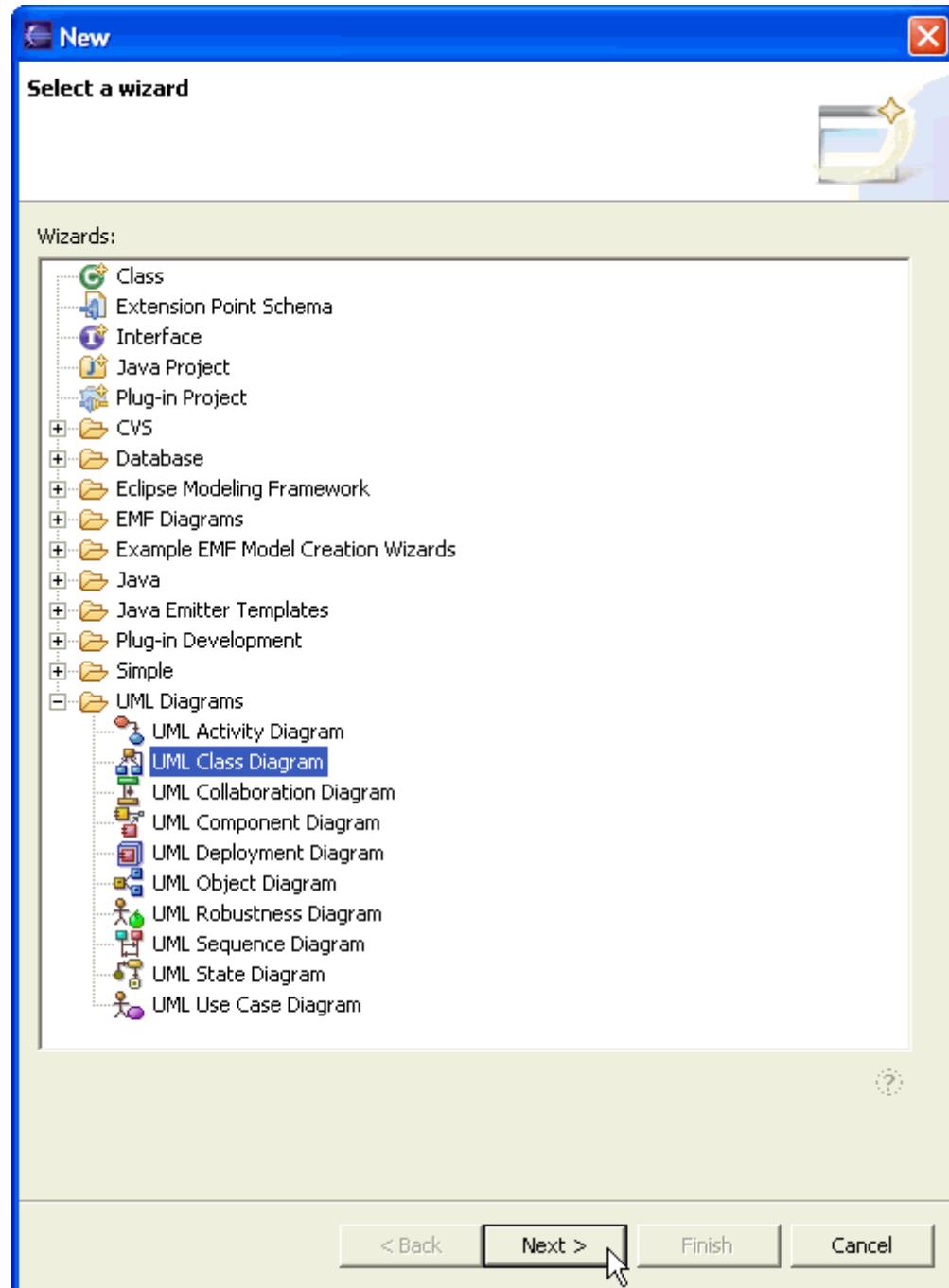
We are going to create a new Java Package.
Select src in the Package Explorer **New > Package**.



Enter the name of the Package.



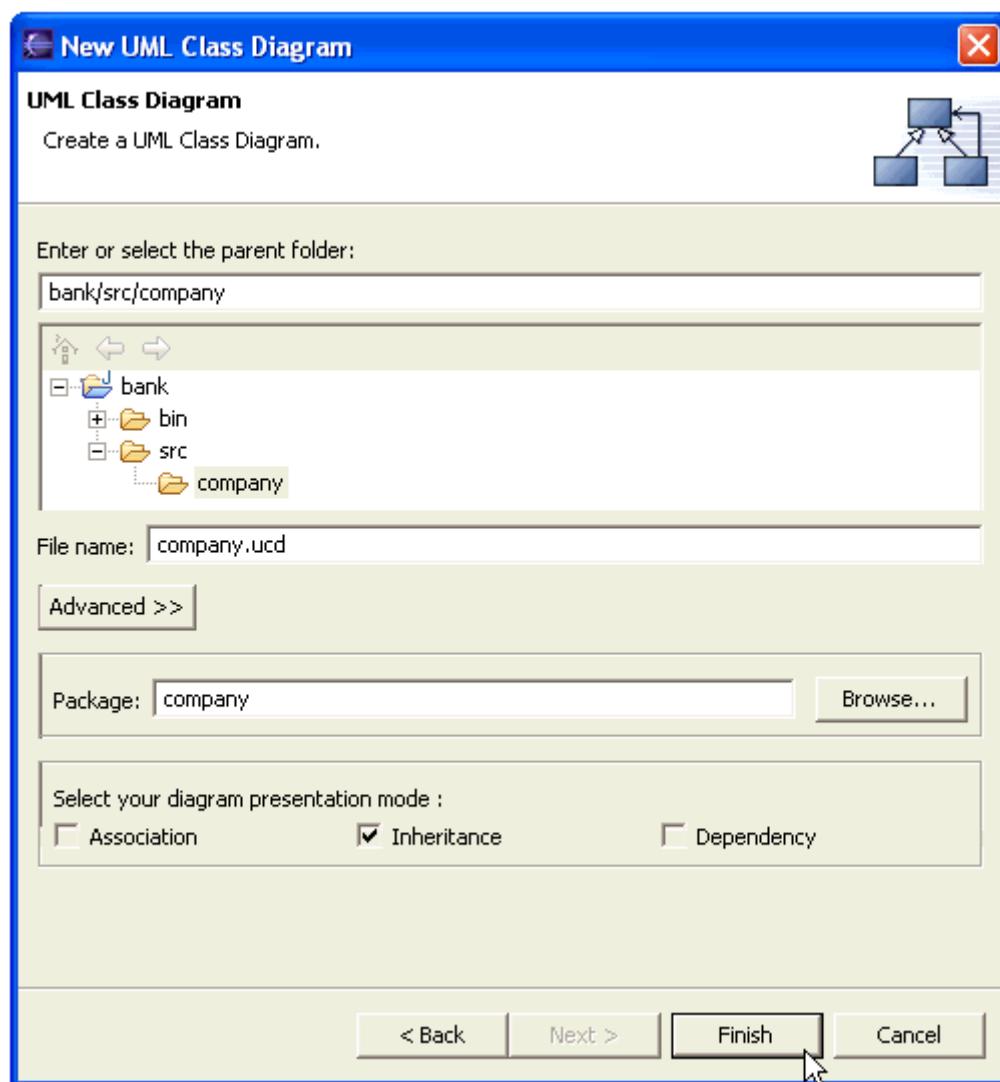
Select company package in the Package Explorer, open the popup menu **New > Other**
Select **UML Diagrams > UML Class Diagram** then click on the next button.



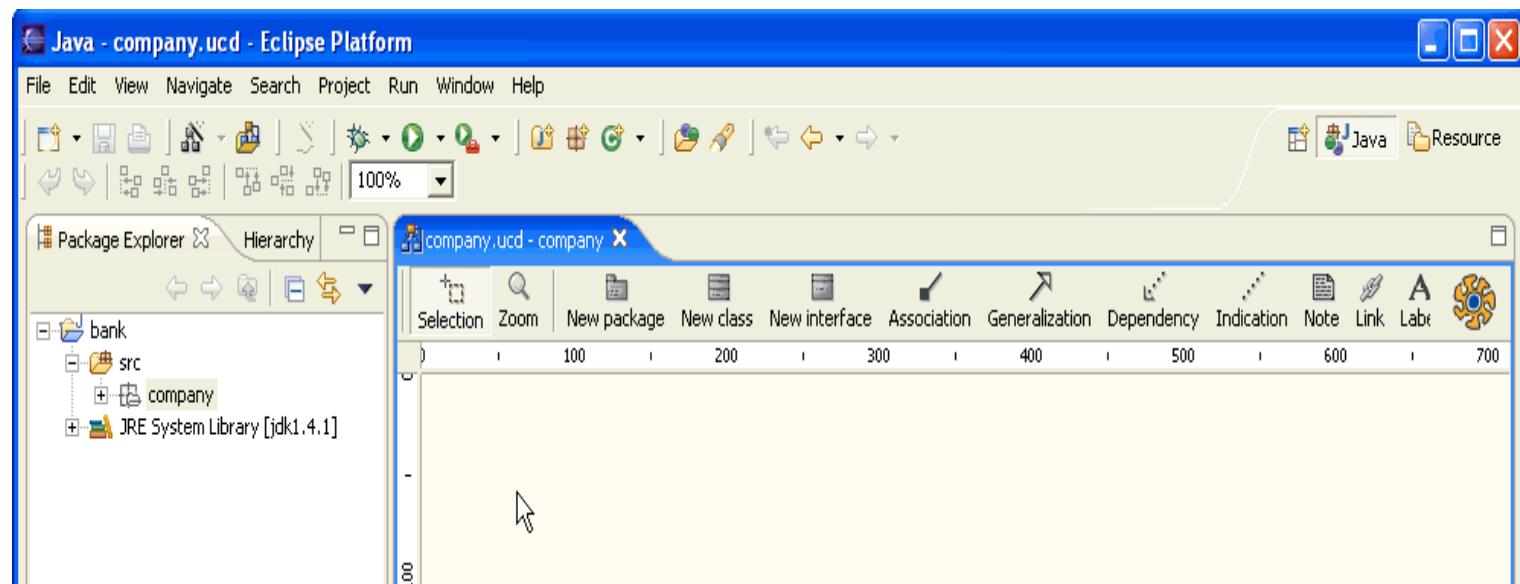
Enter the name of the class diagram file in the File name field.

Enter company, the ucd (UML Class Diagram) extension is related to the class diagram.

Click on the finish button.

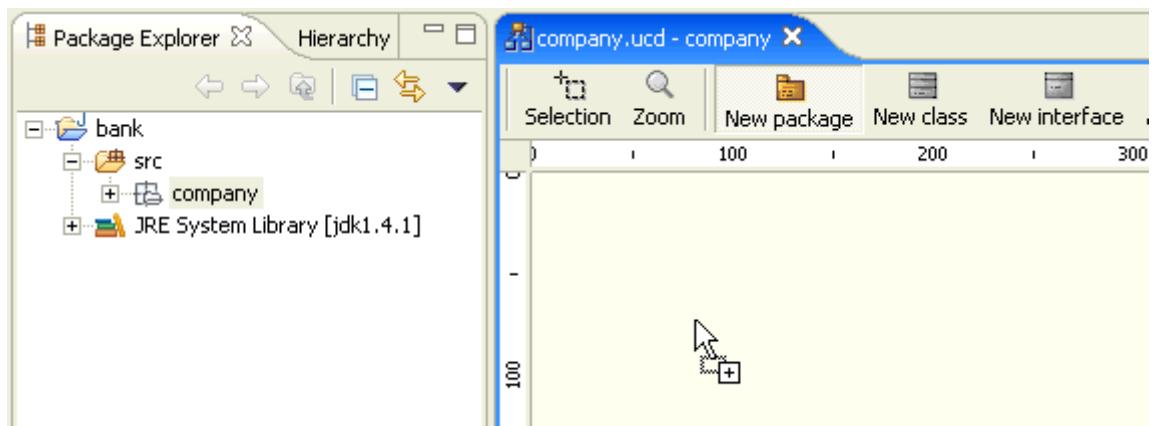


The company class diagram is displayed.

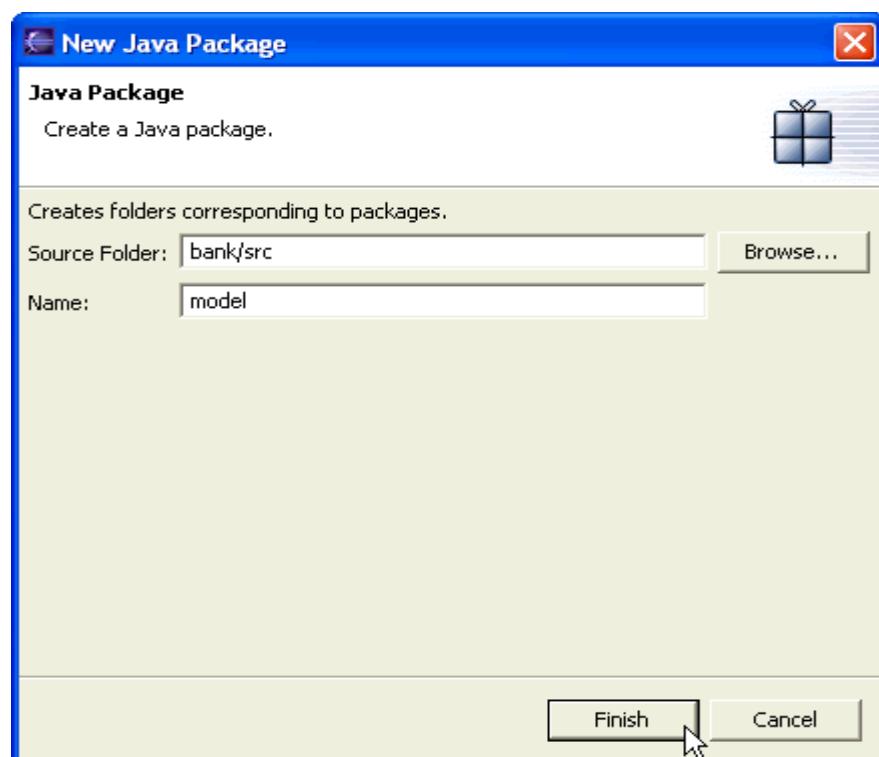


Create a New Package.

Select new package in the toolbar and click on the diagram.

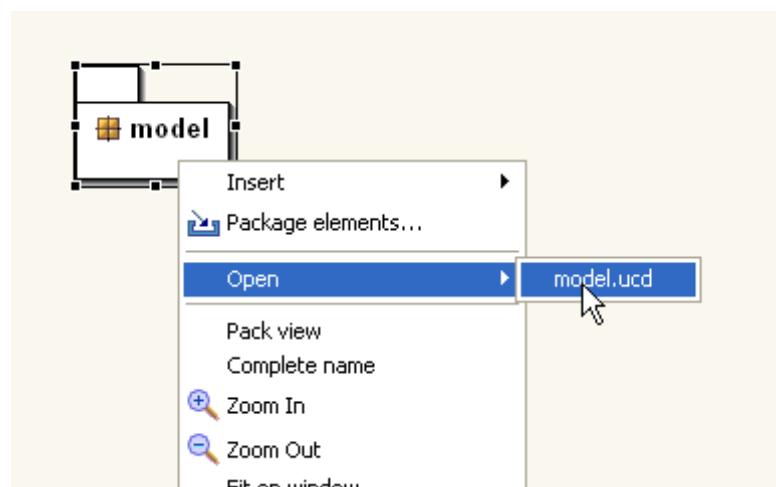


Enter the name of the package in the Name field.



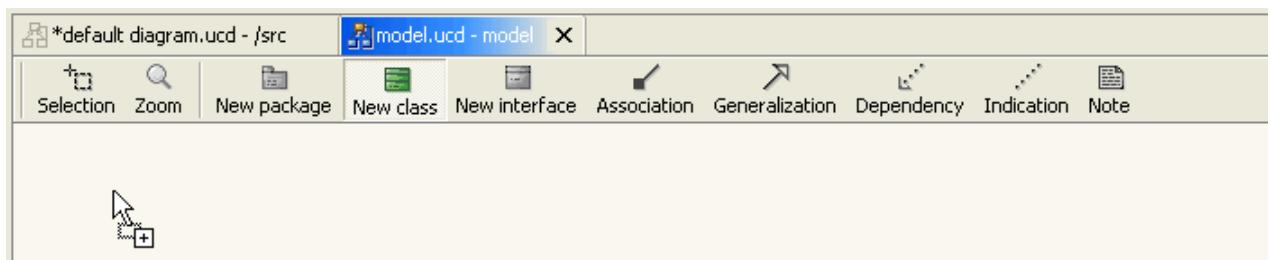
Create a new Class Diagram inside the model Package.

Click on model>Open>model.ucd (ucd is the class diagram extension).

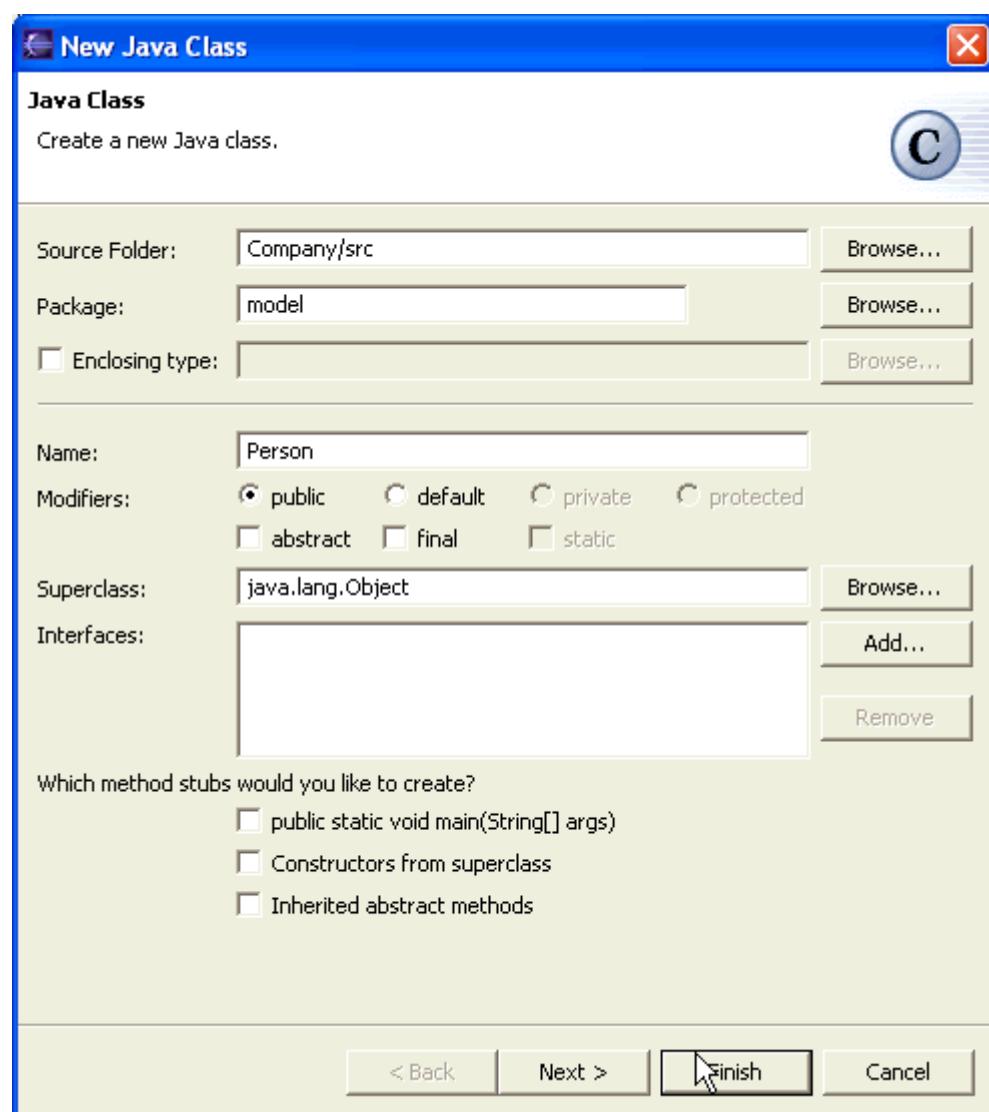


Create a New class.

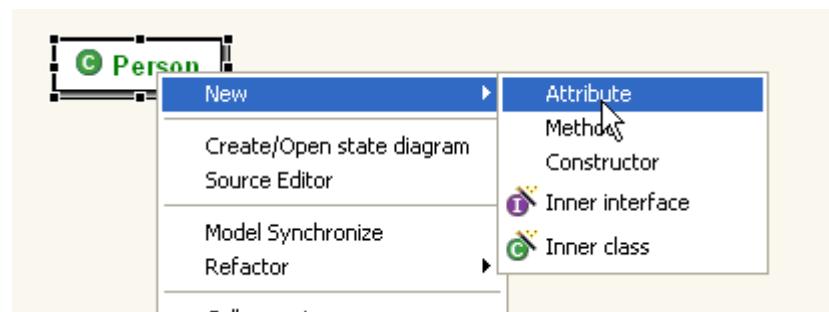
Select New class in the toolbar and click on the diagram.



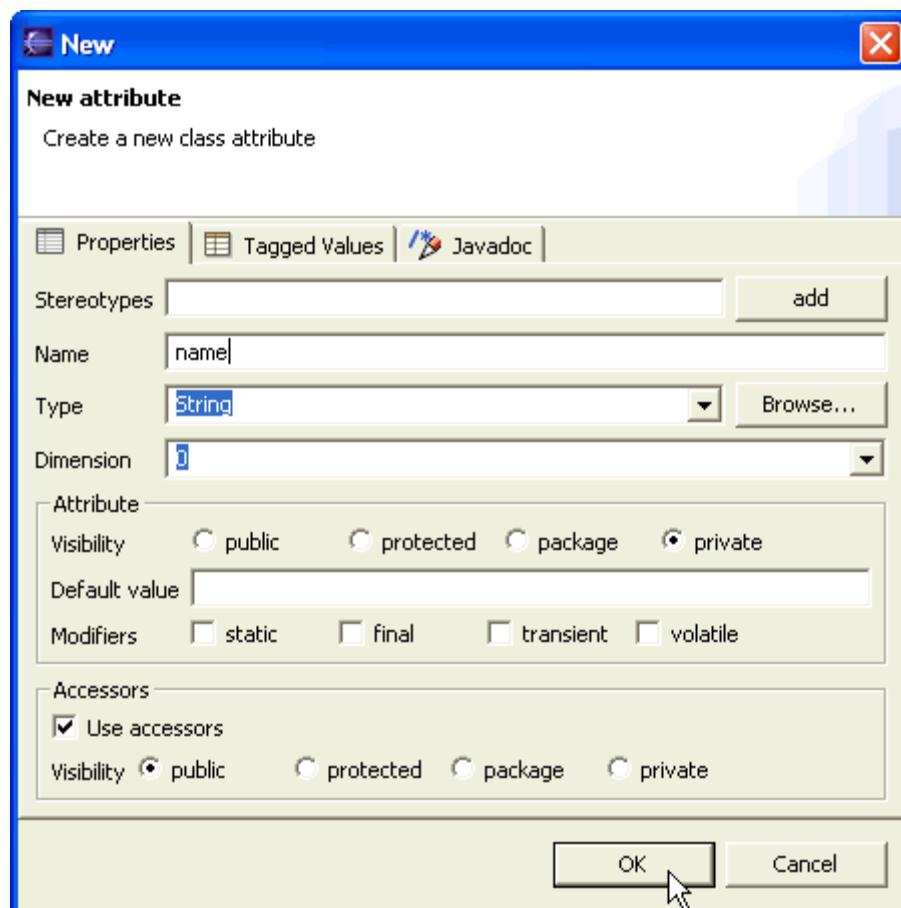
Enter Person and click on the Finish button.



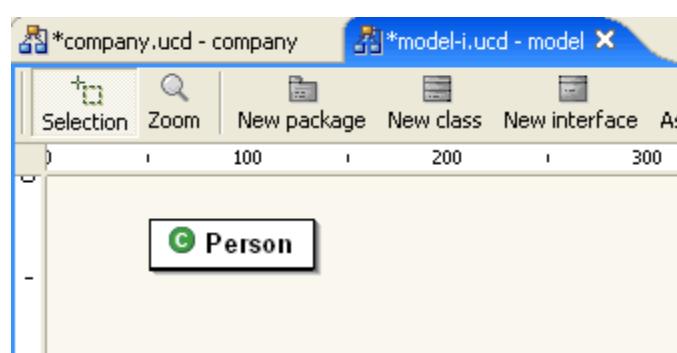
We are going to add two new Attributes.
Click on Person>New>Attribute.



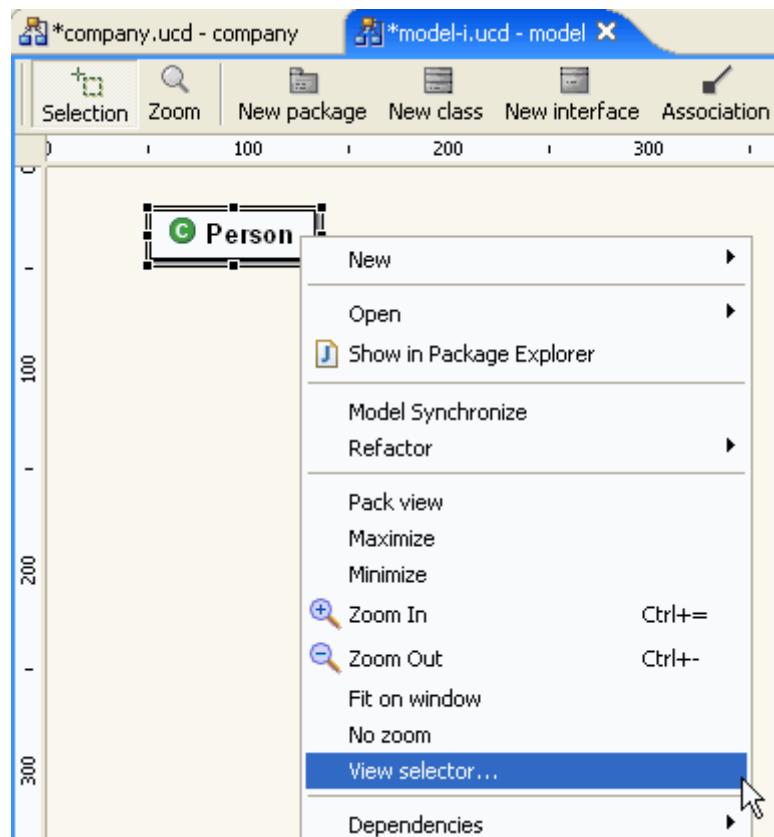
Enter Attributes properties: "age: int" and "name: String".



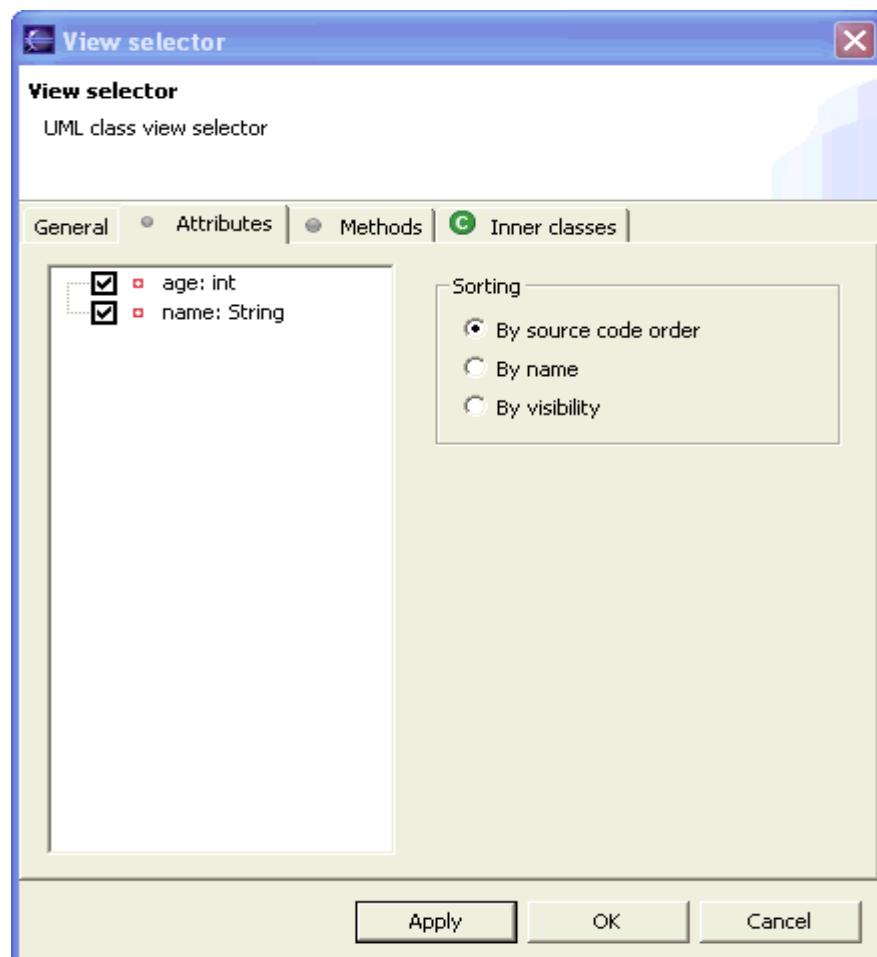
We discover that attributes are not visible in the class diagram editor.



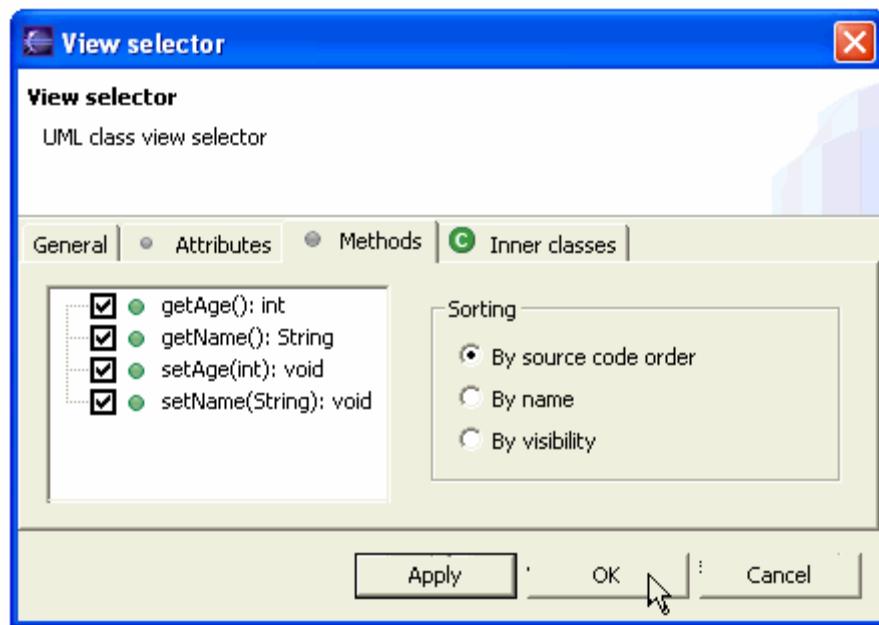
We want to show attributes and Methods in the Class diagram.
Select a class and open the popup menu then select **View selector**.



Select Attributes tab and attributes checkbox in the left pane.



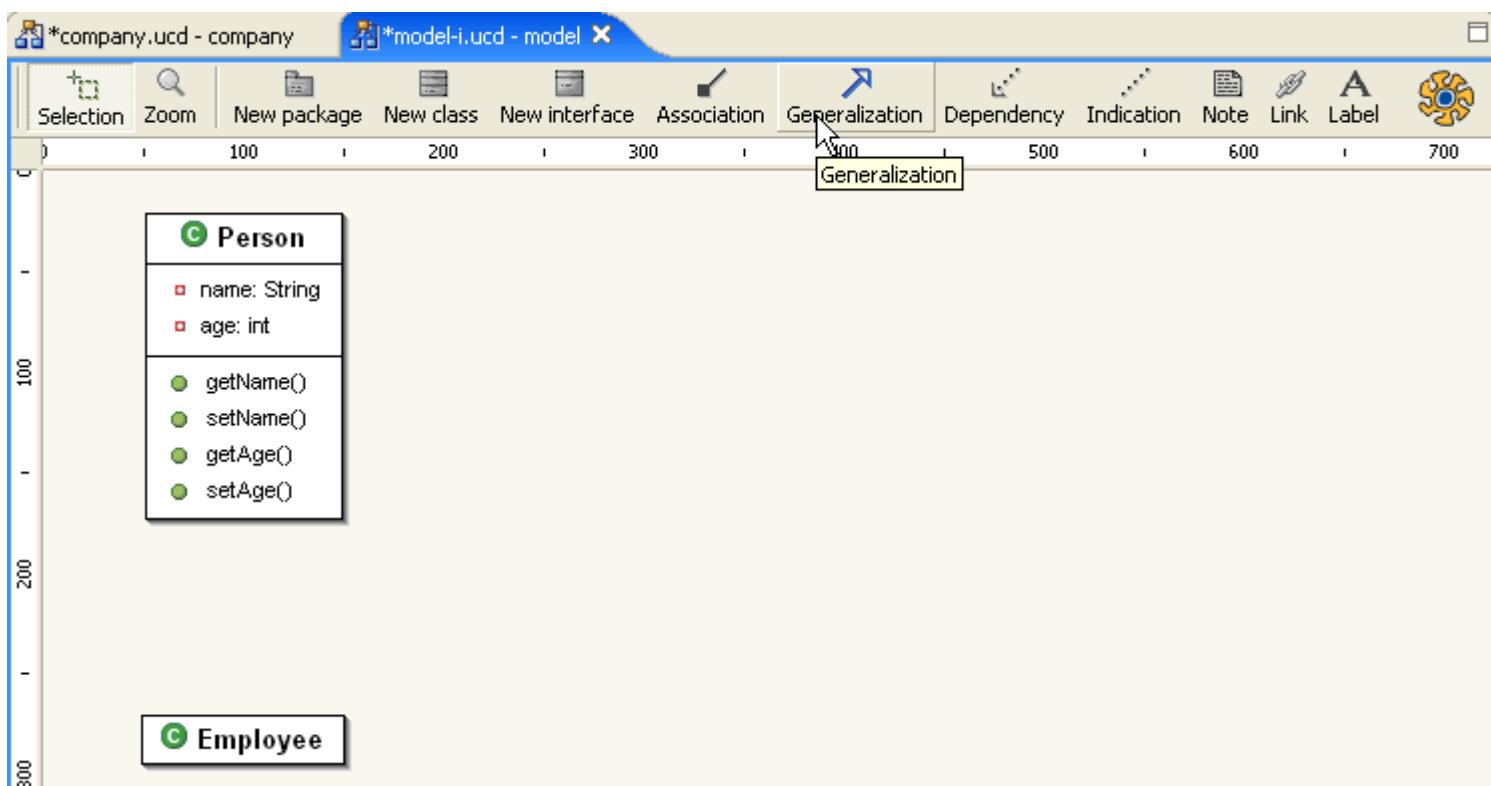
Select Methods tab and Methods checkbox in the left pane.



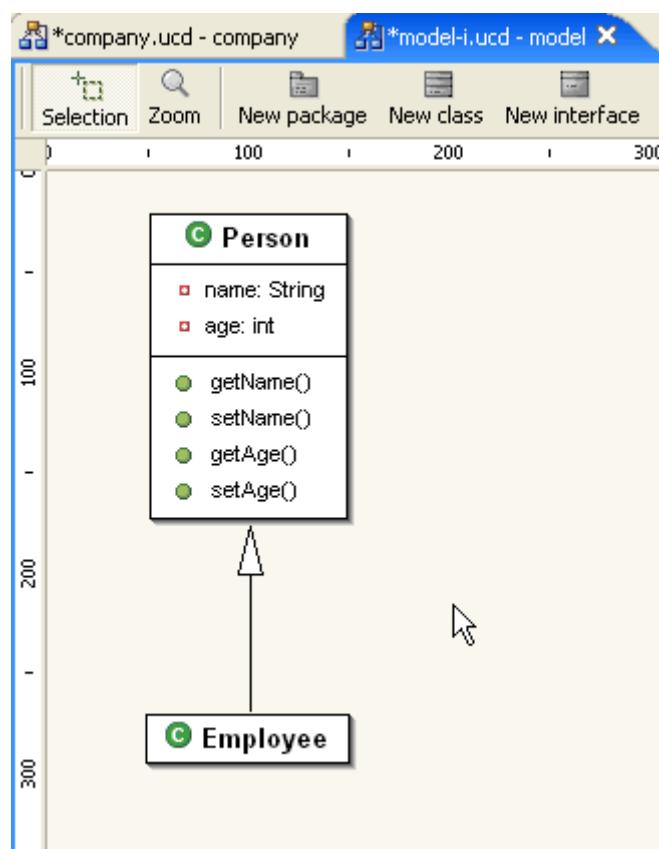
Create a New class named Employee.

We would like to add inheritance from Employee to Person.

Select Generalization in the Toolbar.

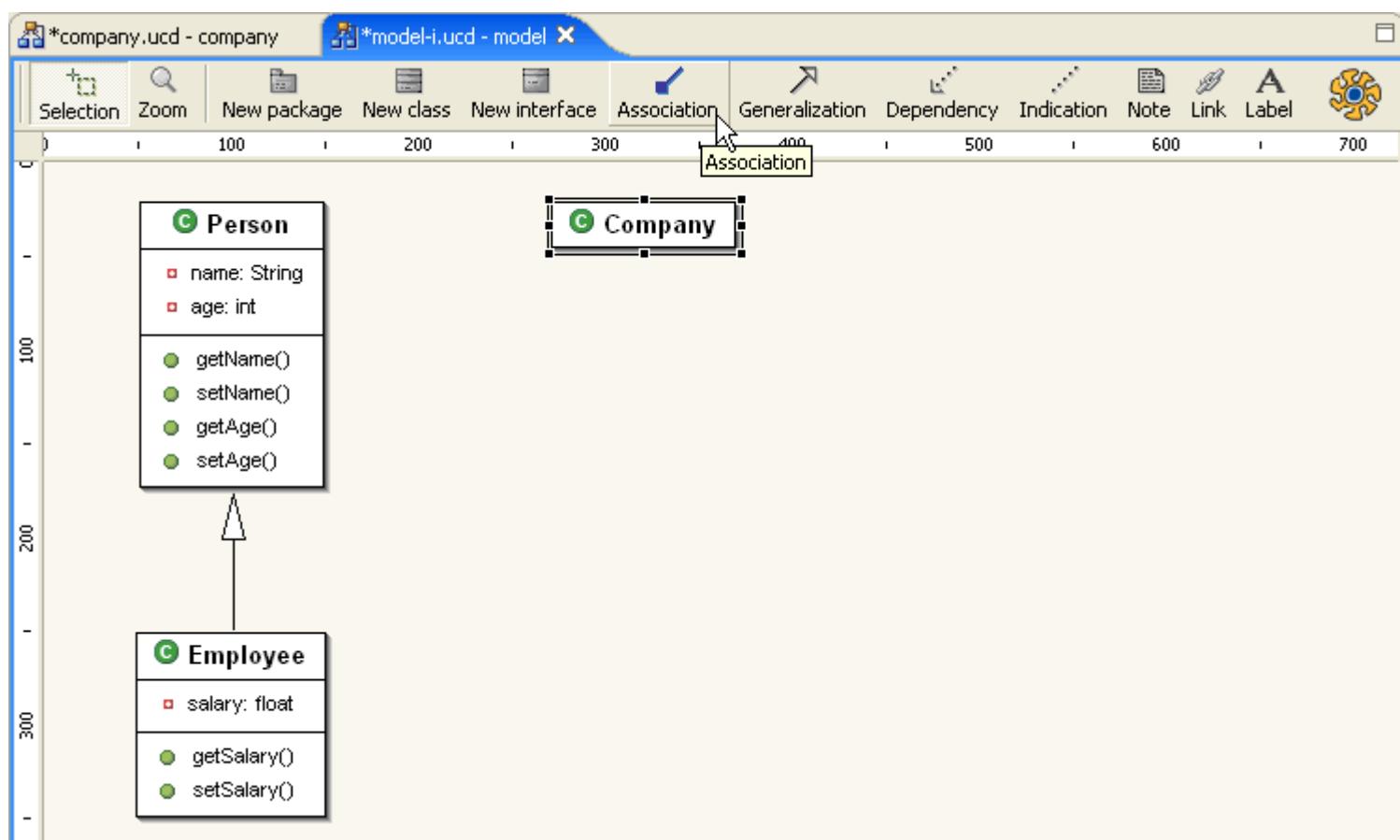


Drag the Inheritance from Employee to Person.

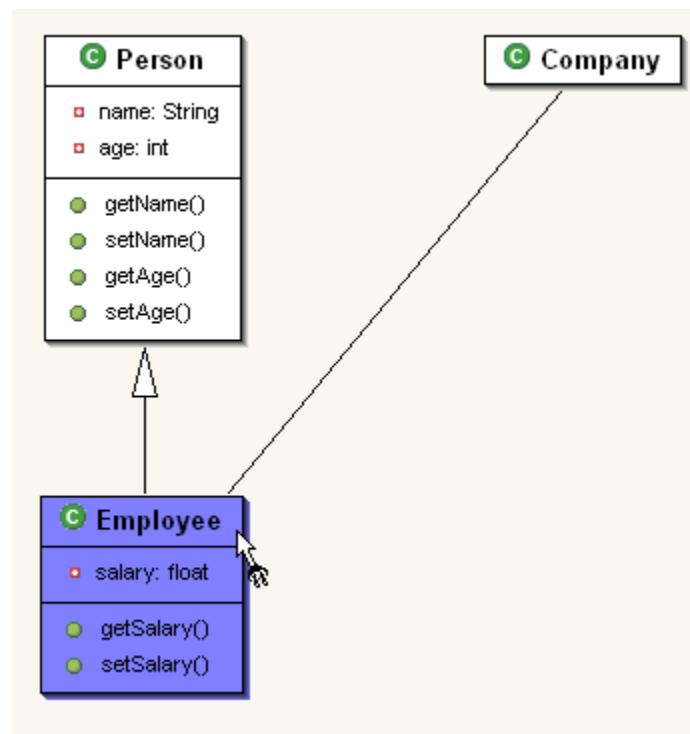


Add "salary: float" attribute to Employee.

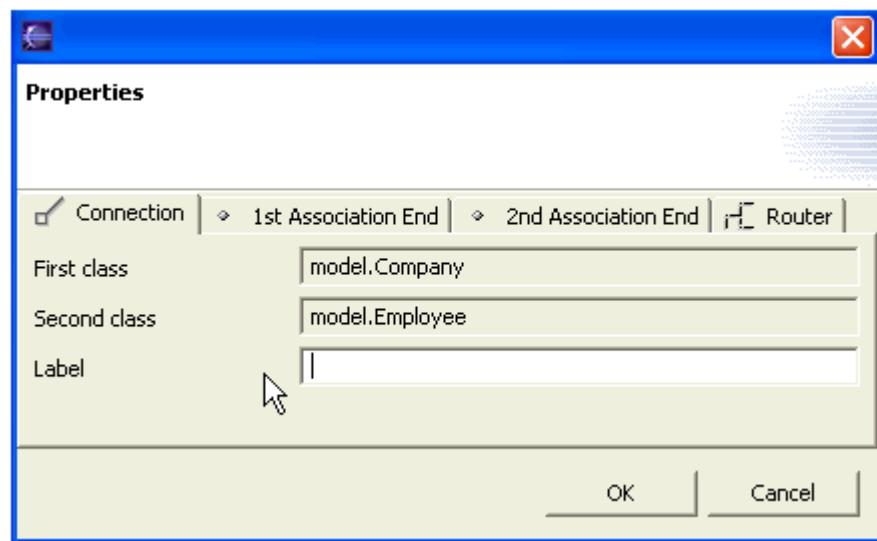
Create a New class named Company.



We want to create a new Association between Company and Employee.
Select Association in the Toolbar then drag from Company to Employee.

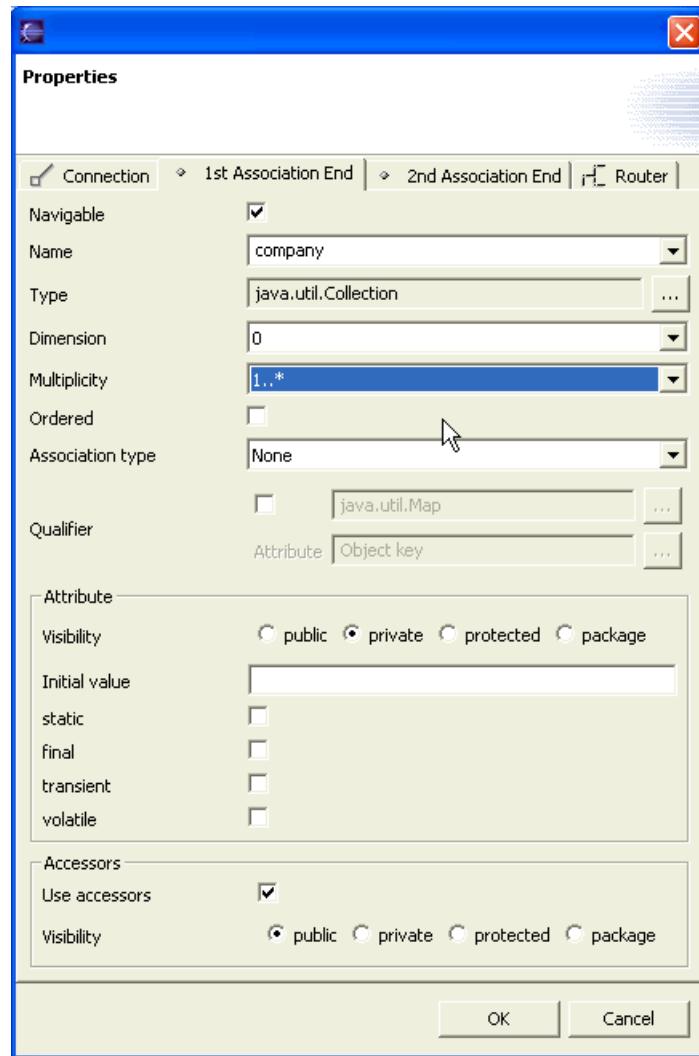


It is possible to add a name to the Label, but we will not in this example.

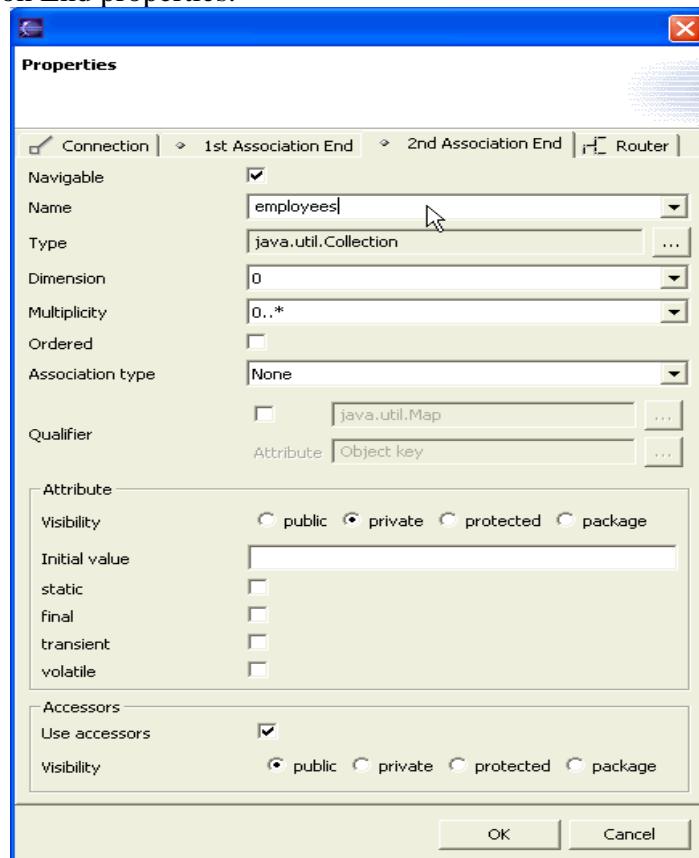


Enter the 1st Association End properties.

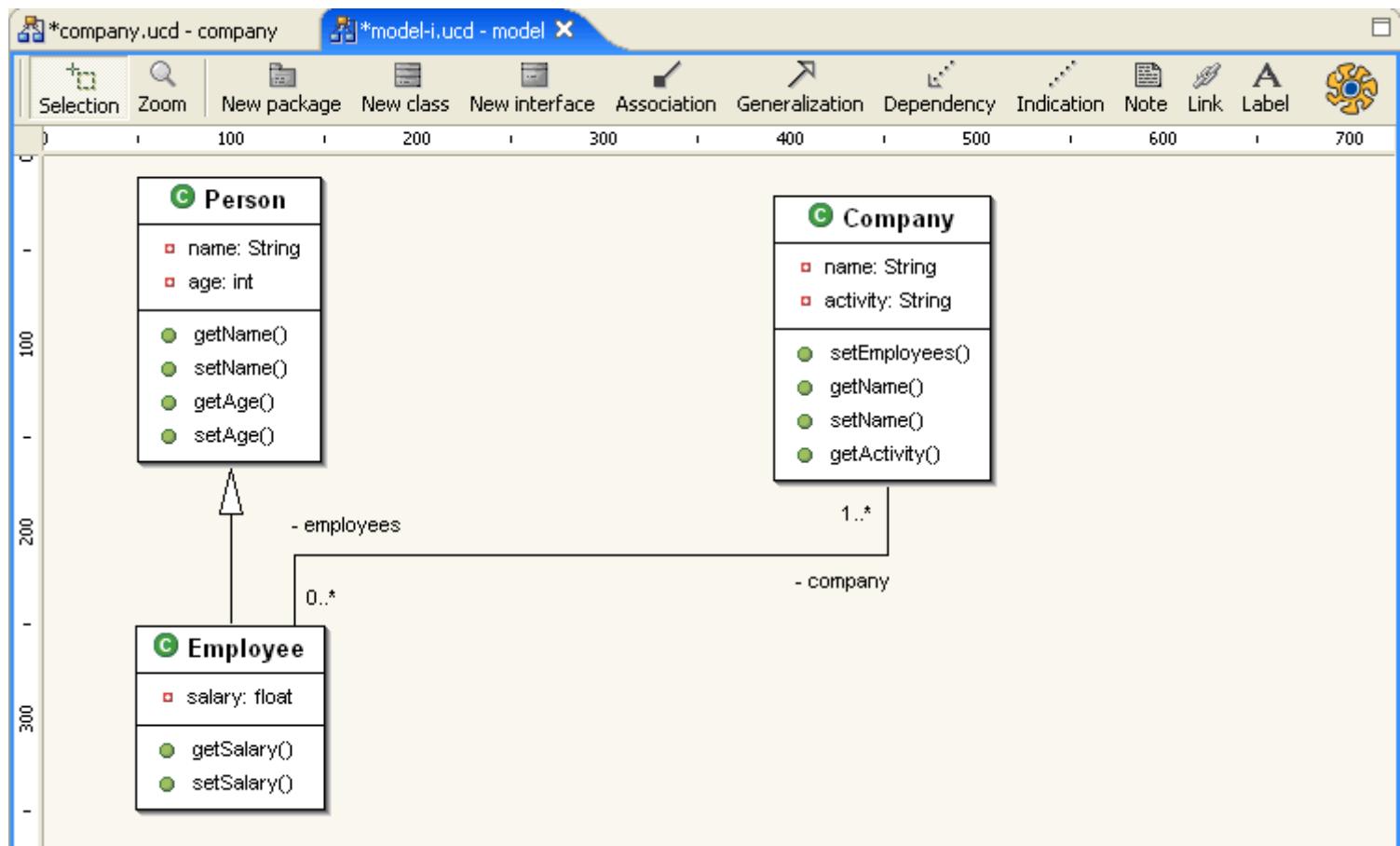
Select 1...* Multiplicity.



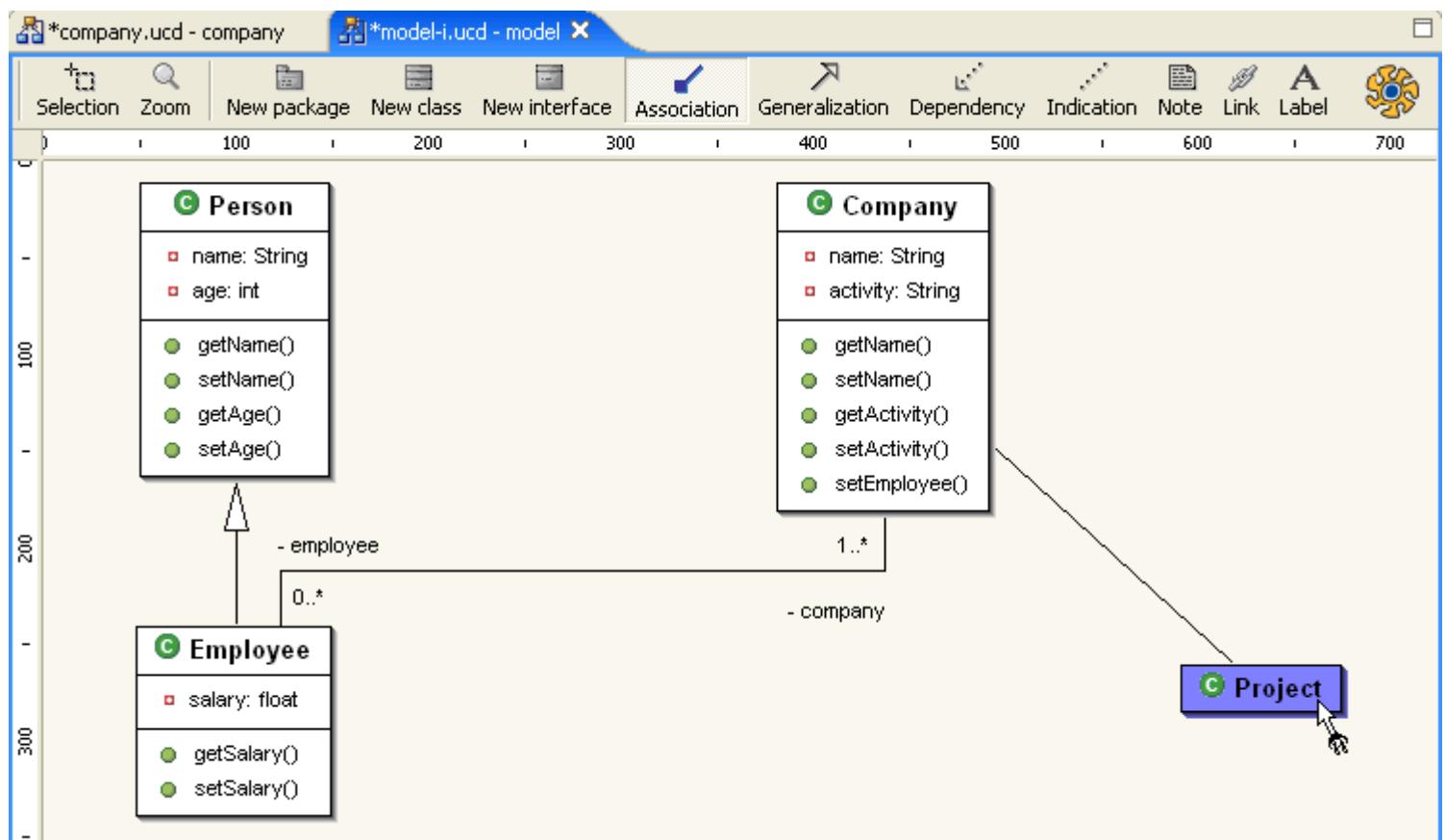
Enter the 2nd Association End properties.



Click on Company and Create two new Attributes: "name: String" and "activity: String".



Create a New class named Project and add an association between Company and Project.



Enter Association Properties.

Label: leave blank.

Properties

<input checked="" type="checkbox"/> Connection	→ 1st Association End	→ 2nd Association End	<input checked="" type="checkbox"/> Router
First class	model.Company		
Second class	model.Project		
Label	<input type="text"/>		

Enter the 1st Association End properties.

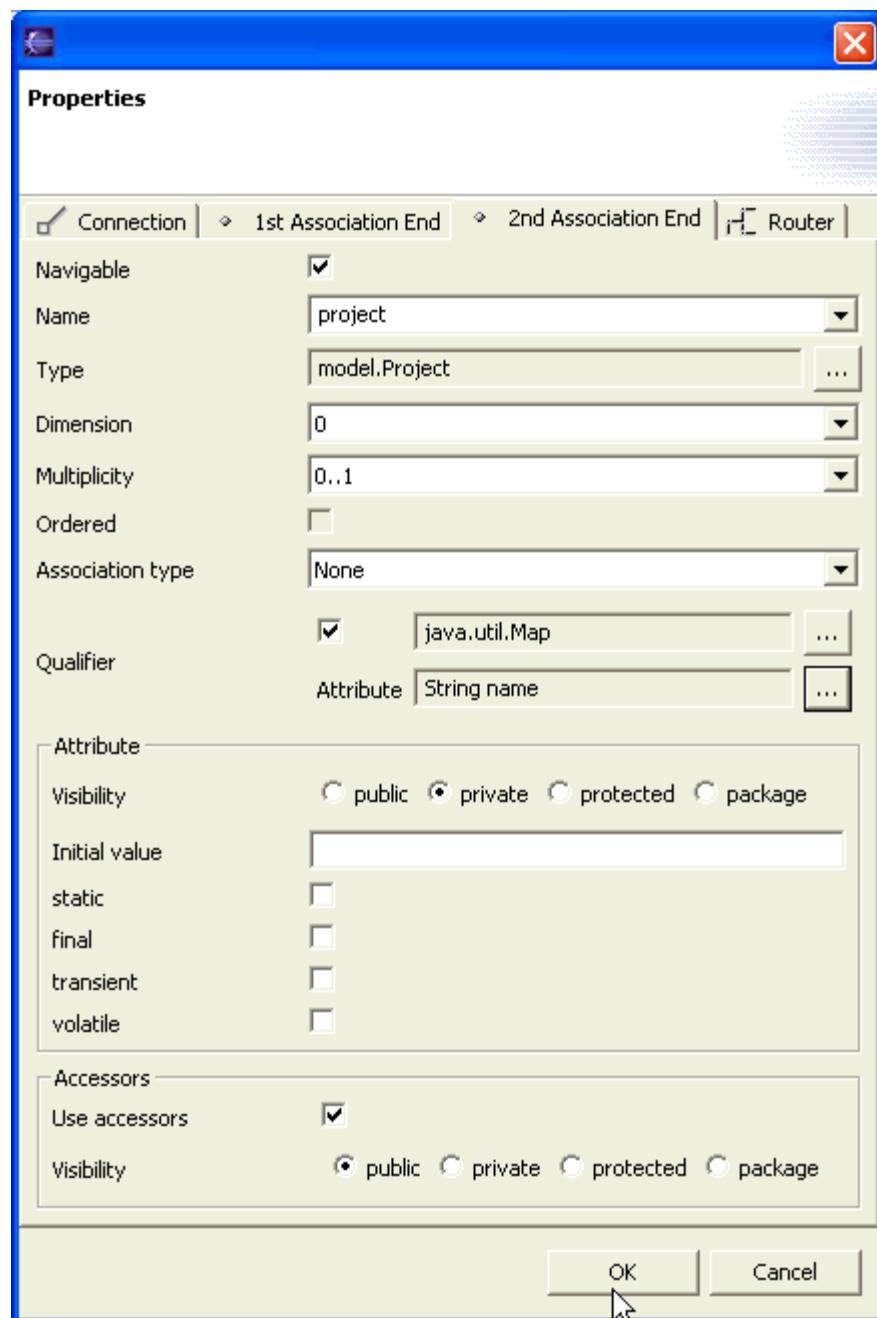
Properties

<input checked="" type="checkbox"/> Connection	→ 1st Association End	→ 2nd Association End	<input checked="" type="checkbox"/> Router
Navigable	<input checked="" type="checkbox"/>		
Name	company		
Type	model.Company		
Dimension	0		
Multiplicity	1		
Ordered	<input type="checkbox"/>		
Association type	None		
Qualifier	<input type="checkbox"/>	java.util.Map	<input type="button"/>
	Attribute	Object key	<input type="button"/>
Attribute			
Visibility	<input type="radio"/> public <input checked="" type="radio"/> private <input type="radio"/> protected <input type="radio"/> package		
Initial value	<input type="text"/>		
static	<input type="checkbox"/>		
final	<input type="checkbox"/>		
transient	<input type="checkbox"/>		
volatile	<input type="checkbox"/>		
Accessors			
Use accessors	<input checked="" type="checkbox"/>		
Visibility	<input checked="" type="radio"/> public <input type="radio"/> private <input type="radio"/> protected <input type="radio"/> package		
<input type="button"/> OK <input type="button"/> Cancel			

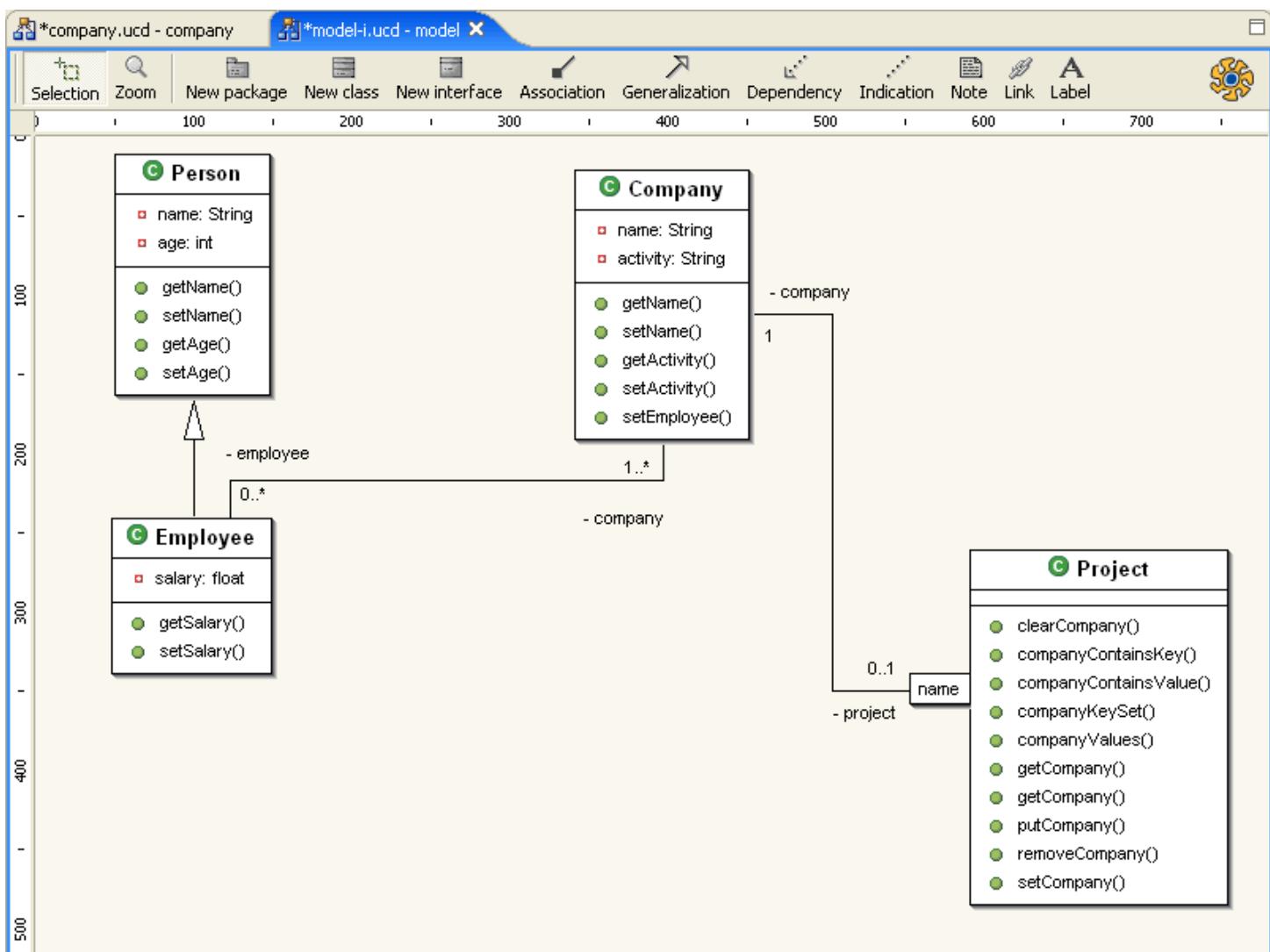
Enter the 2nd Association End properties.

Select Qualifier in the 2nd Association End.

Select the Qualifier checkbox and enter name in the name field and Attribute String name.

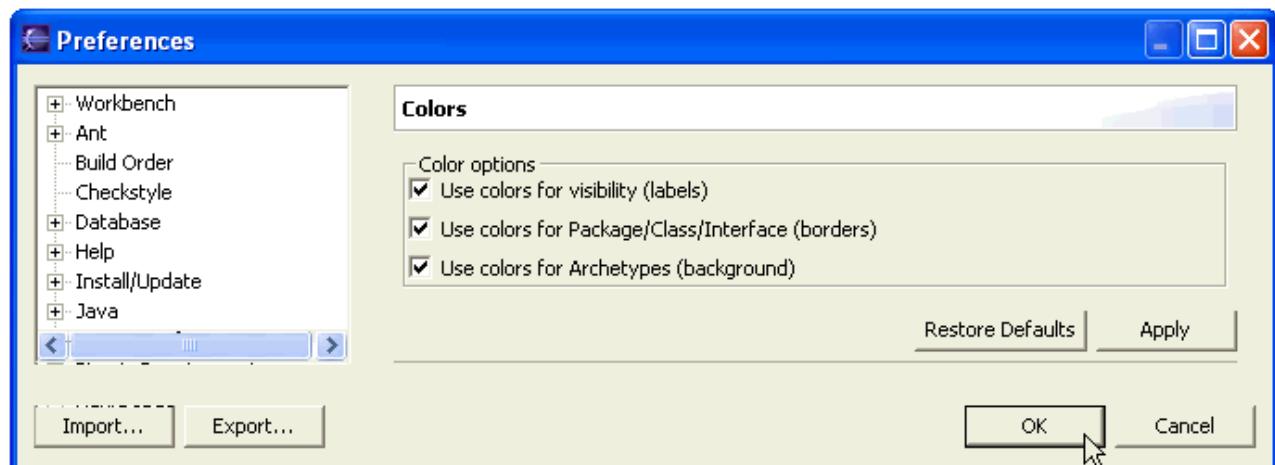


Please check that your Class diagram is the same as below.

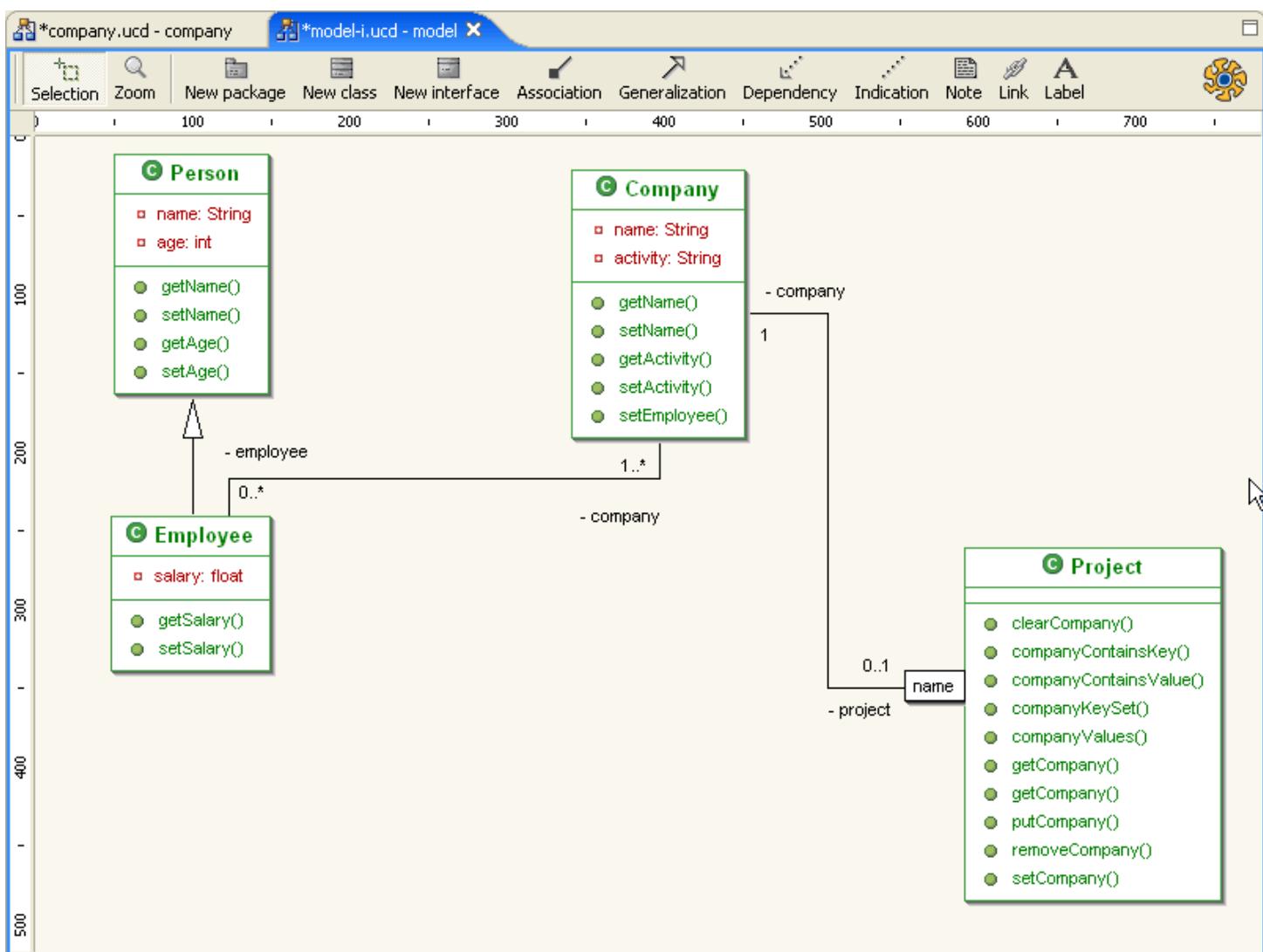


We can use colors to have a better understanding of our diagram.

Select in the menu bar Windows > Preferences > UML > Class Diagram > Colors

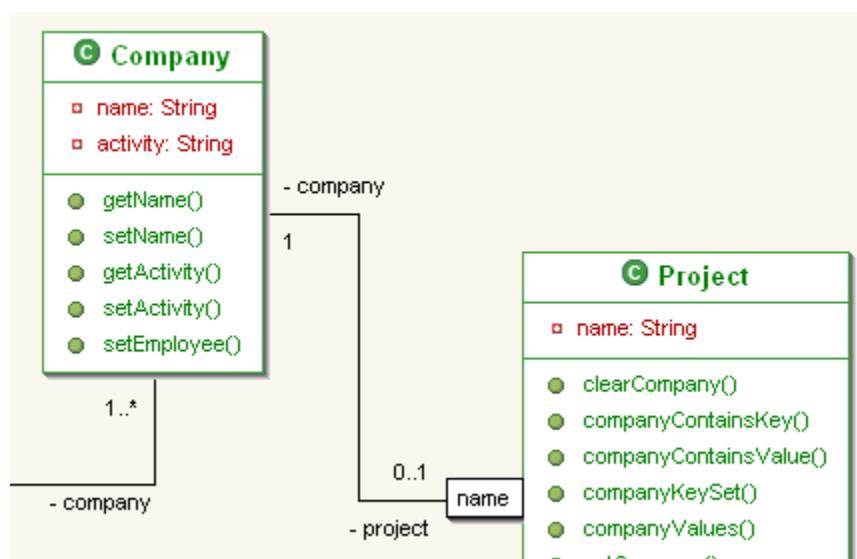


Colors have been included in the class diagram below.



Add "name: String" new Attribute to Project class.

Use the view selector to hide getters and setters.



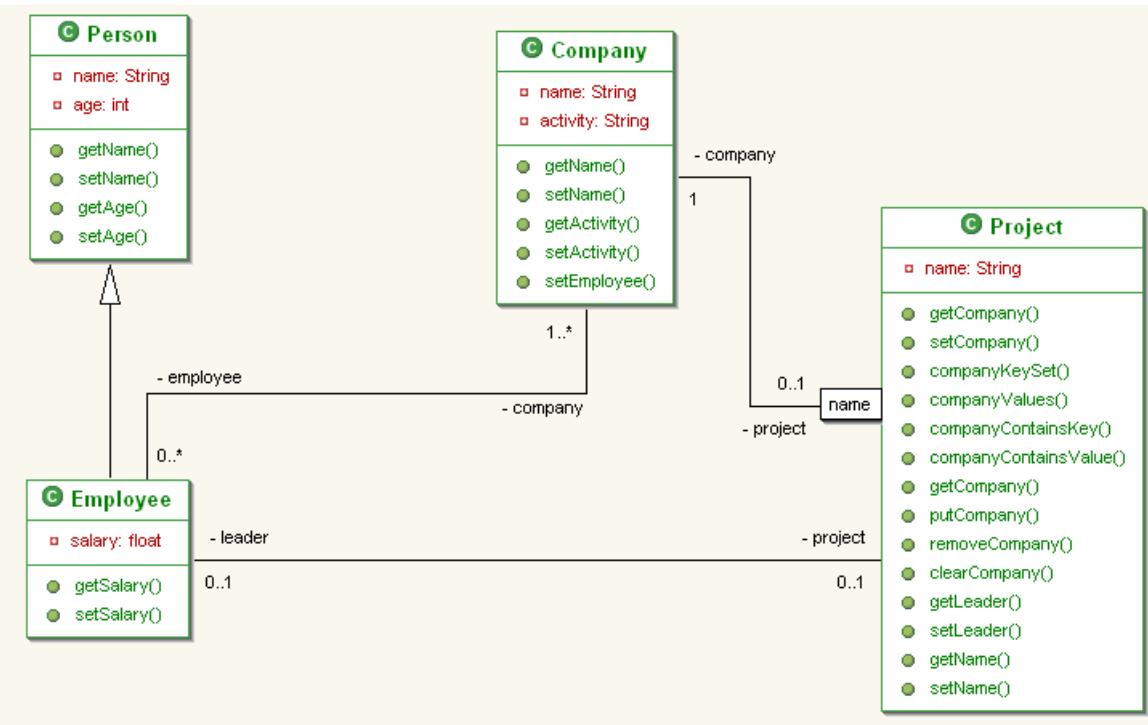
Add an Association between Employee and Project.
Select Association and drag from Employee to Project.

In the properties menu:

Connection: leave blank

1st Association End: Name: leader, Dimension 0, Multiplicity 0..1

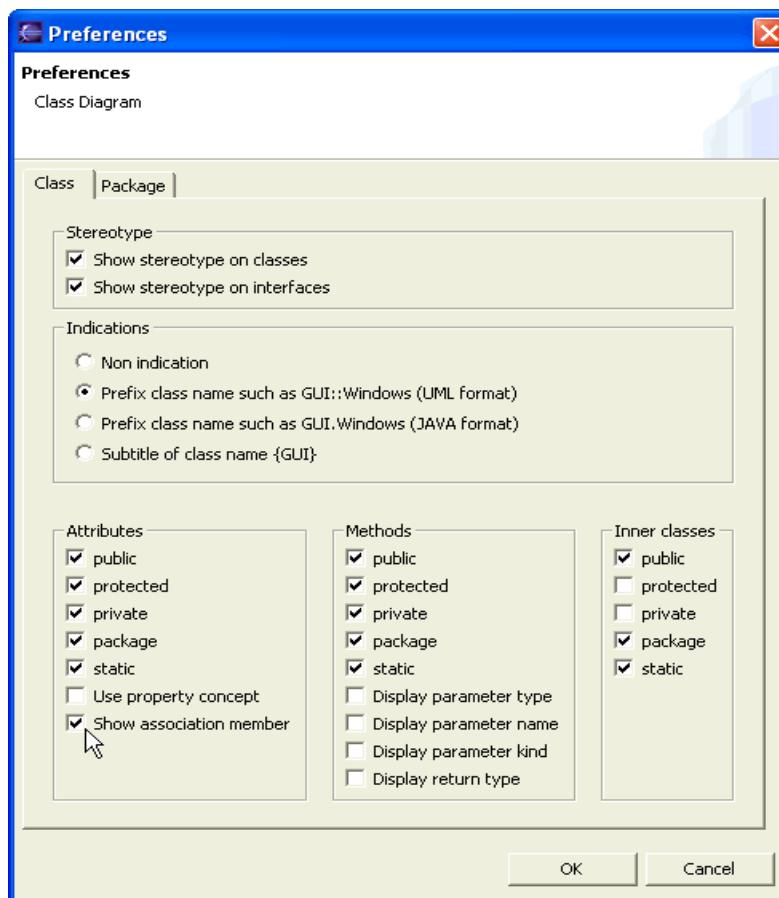
2nd Association End: Name Project, Dimension 0, Multiplicity 0..1



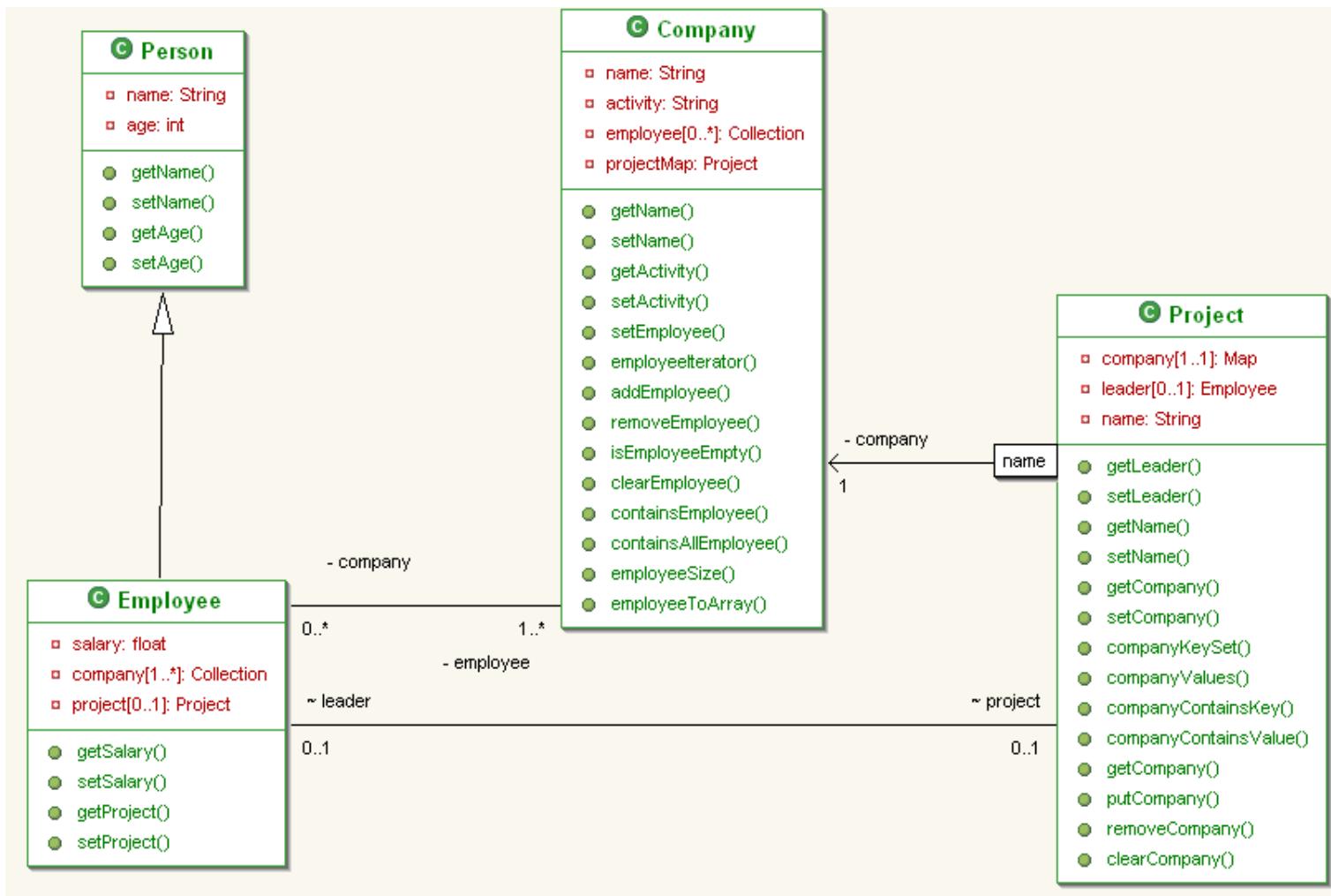
We also need to show association members.

Right click inside the diagram editor, making sure that you don't select one of the elements - otherwise you are going to open the element popup menu and not the diagram popup menu.

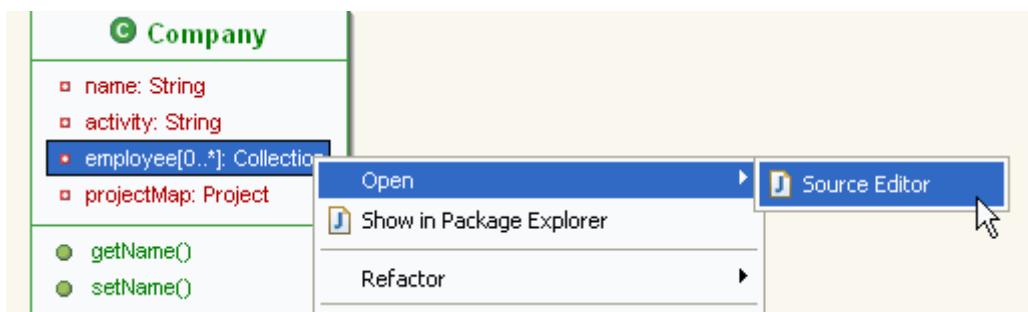
Select the Show association member checkbox and click on the OK button.



All association attributes are now displayed in the class diagram.



Select an element and right click to open the **element popup menu** > **View Selector**
 Use view selector (if needed) to display employee, activity and projectmap attributes etc...
 Show code. Click on **Employee attribute**>**Open**>**Source Editor**



Type new code:

```
private Collection employees = new ArrayList();
```

```
private Map projectMap = new Hashtable();
```

Copy Java code inside company class.
The marker shows unresolved type.

The screenshot shows a Java code editor window with a file named *Company.java. The code contains a method that iterates over a map of projects. At the line `Iterator iterator = projectMap.values().iterator();`, there is a yellow warning marker. A tooltip appears over the word `Iterator`, showing the message "Unresolved type: Iterator". The code editor interface includes tabs for other files like *Employee.java and *Project.java, and various toolbars and status bars.

```

    }
    if (employees != null)
    {
        Iterator iterator = projectMap.values().iterator();
        while (iterator.hasNext())
        {
            Project element = (Project) iterator.next();
            buffer.append(" Project: " + element.getName() + "\n");
        }
    }

    return buffer.toString();
}

```

Correct the compilation error -> Resolve the type.
Place the mouse cursor at the end of "Iterator".
Control + space and select Iterator - java.util

The screenshot shows the same Java code editor window. The mouse cursor is now placed at the end of the word `Iterator` in the line `Iterator iterator = projectMap.values().iterator();`. A detailed tooltip has appeared, providing information about the `Iterator` class from the `java.util` package. The tooltip includes the following text:
An iterator over a collection. Iterator takes the place of Enumeration in the Java collections framework. Iterators differ from enumerations in two ways:

- Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.

See Also:
[Collection](#)
[ListIterator](#)
[Enumeration](#)
[@author](#)

Now you can go back to your class diagram and enjoy modeling!

The class diagram extension is *.ucd.

Sequence Diagram

Concept

A sequence diagram is a model describing how groups interact over time. It is an interaction diagram that details how operations are carried out: what messages are sent and when.

You use the sequence diagram model to describe the aspects of your system that change over time. These are events that mark changes, sequences of events and so forth. The time sequence interaction is shown in the sequence diagram and consists of two dimensions. The vertical dimension (time) and the horizontal dimension (different objects).

UML sequence diagrams can be used to:

- explore your design
- feel which classes are going to be complex
- validate the logic and completeness of a usage scenario
- detect bottlenecks within an object-oriented design

The elements of the sequence diagrams are available in the tool bar:

 New Object can be created by selecting the icon in the toolbar. Allows you to add a new instance within the diagram.

 New Actor can be created by selecting the icon in the toolbar

 Message can be created by selecting the icon in the toolbar. Each message represents a method call.

 Self Message can be created by selecting the icon in the toolbar. A reflexive message is a message sent by an instance to itself.

Drag and Drop

In a sequence diagram, you can drag and drop a class or an interface from the Package Explorer to the Sequence Diagram Editor and get a new instance.

Toolbar Items

Actor

Add a new actor on the diagram.

Instance

Add a new instance on the diagram.

Message

This tool creates an interaction between an actor/instance and an instance. Each message represents a method call.

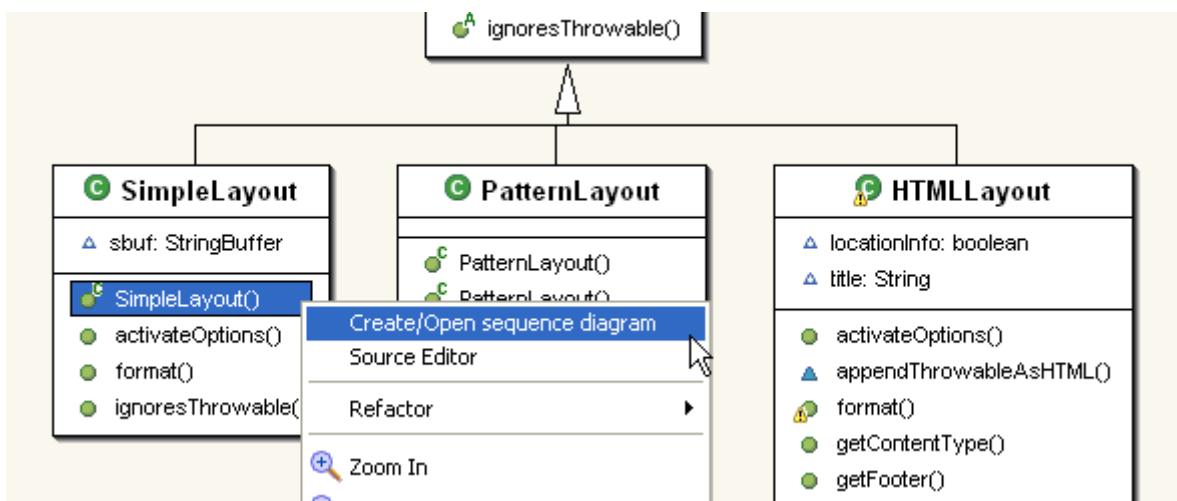
Self Message

This tool allows you to create reflexive messages. A reflexive message is a message sent by an instance to itself

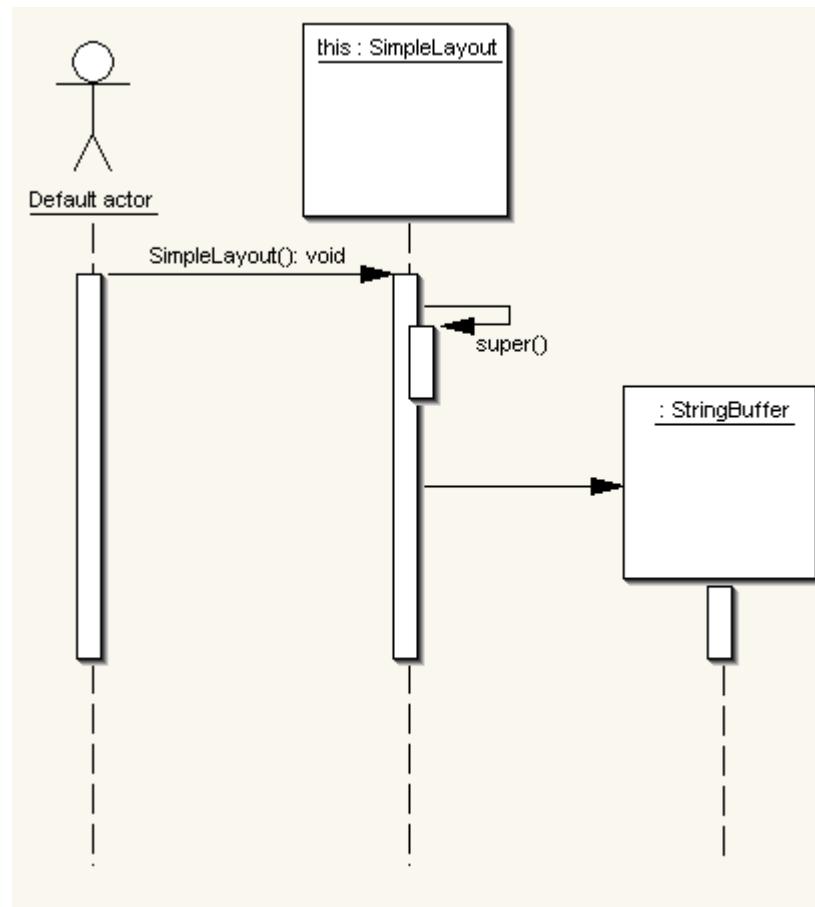
Reverse Engineering

Double click on a method inside the class diagram editor or select a method in the package explorer **open the class diagram popup menu > Create/open sequence diagram**.

The following example is the reverse of the SimpleLayout() method in the SimpleLayout class from the org.apache.log4j.or package.



This is the result of the reverse of the SimpleLayout() method.



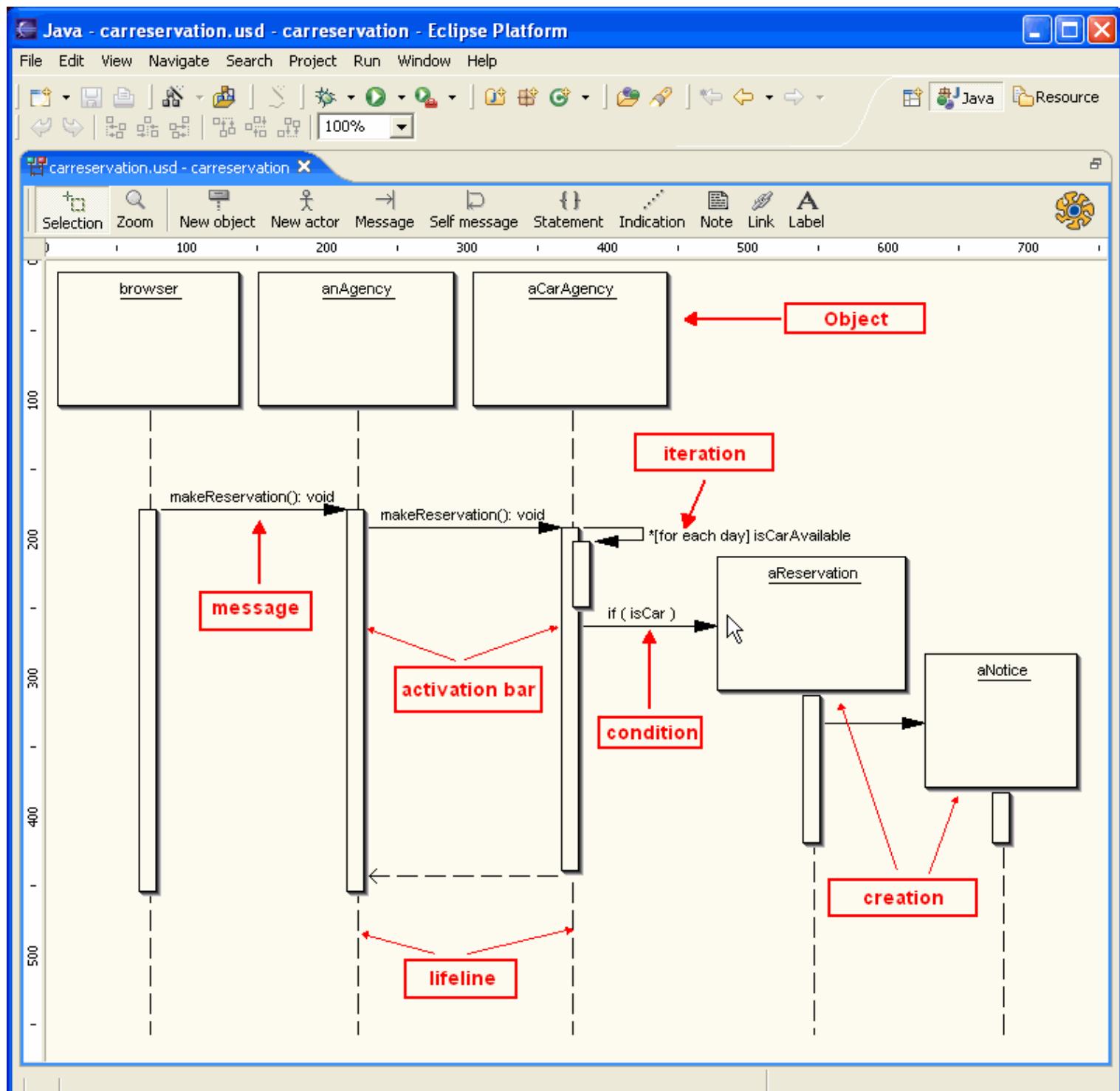
Sequence Diagram Example

Below is a sequence diagram for making a car reservation using a browser.

The object initiating the sequence of message is a reservation internet browser.

The internet browser sends a makeReservation() message to a travel agency, which sends a makeReservation() message to a car rental reservation. If the car is available, then it makes a Reservation and a Confirmation. Each vertical line represents the time that an object exists. Each arrow is a message call. An arrow goes from the sender of the activation bar of the message on the receiver's lifeline. The activation bar represents the duration of execution of the message.

The car rental reservation issues a self call to determine if a car is available. If so, the car rental reservation makes a Reservation and a Confirmation.



The file deployment diagram extension is *.usd.
Notes can be included.

UseCase Diagram

Concept

The Use Case Diagram describes the behavior of a system from a user's standpoint : functional description and its major processes.

It provides a graphic description of who will use a system and what kind of interactions to expect within that system. Use Case Diagrams are useful when describing requirements for a system in the analysis, design, implementation and documentation stages.

A Use Case Diagram is a collection of actors, Use Cases, and their communications.

The elements of the Use Case diagrams are available in the tool bar:

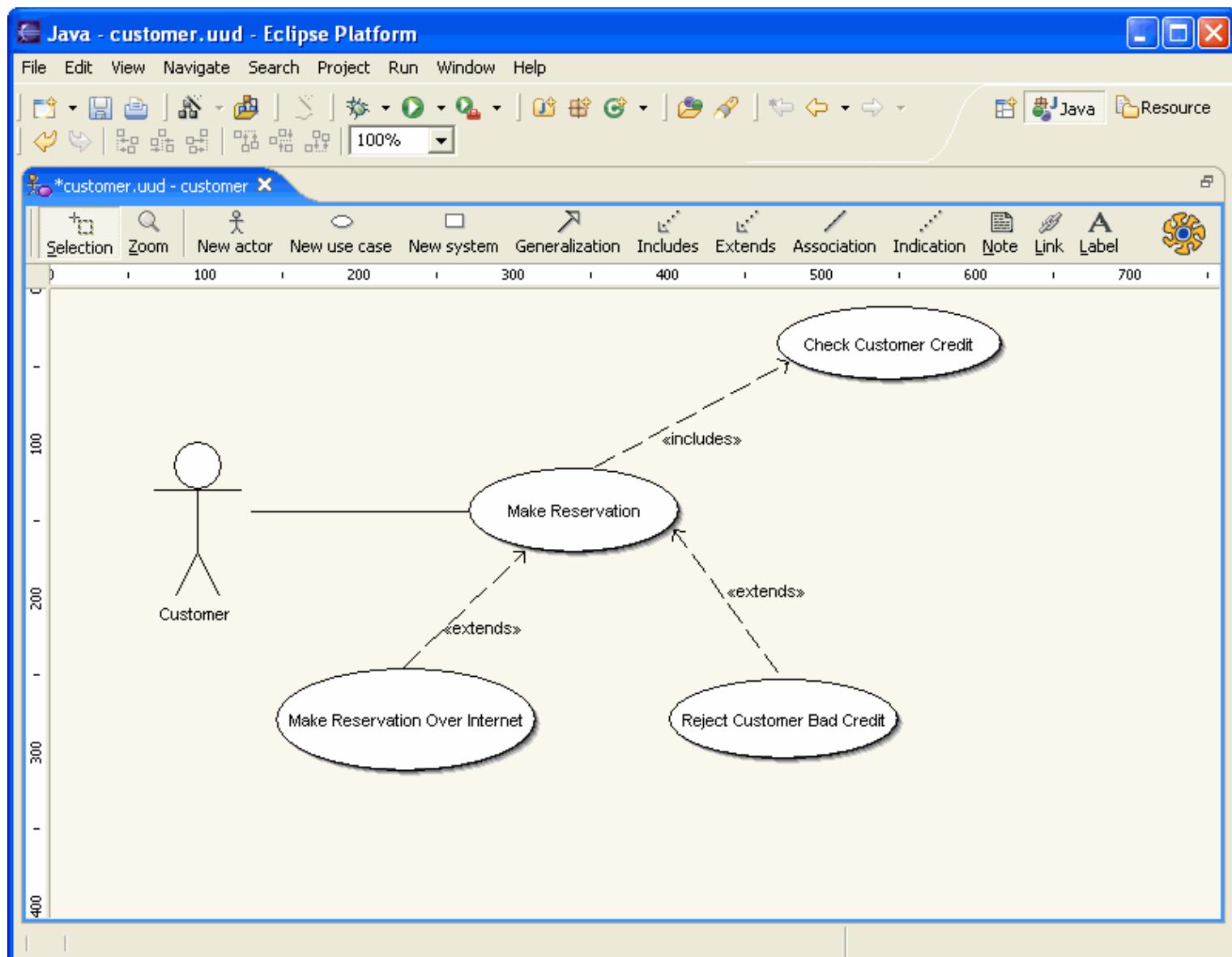
- 👤 New Actor can be created by selecting the icon in the toolbar.
- ⭕ New Use Case can be created by selecting the icon in the toolbar.
- ◻ New system can be created.
- ↗ Create inheritance relationships between Use Cases or actors.
- ↖ The Includes Relationship is used to indicate that the source Use Case includes the behavior of the target Use Case.
- ↙ The Extends Relationship indicates that the source Use Case adds its behavior to the target Use Case.
- ↙ Association is linking an actor to its Use Cases.

UseCase Diagram Example

Below is an example of the Use Case as part of a diagram with one actor and four Use Cases.

In this example the Customer is an actor who Makes a Reservation.

[See also the Robustness Diagram example.](#)



The file deployment diagram extension is *.uud.

State Diagram

Concept

A state diagram shows the sequences of states that an object or an interaction goes through during its life in response to received stimuli, together with its responses and actions. The state of an object depends on its current activity or condition. A state diagram shows the possible states of the object and the transition that cause a change in state.

State chart diagrams are often used to

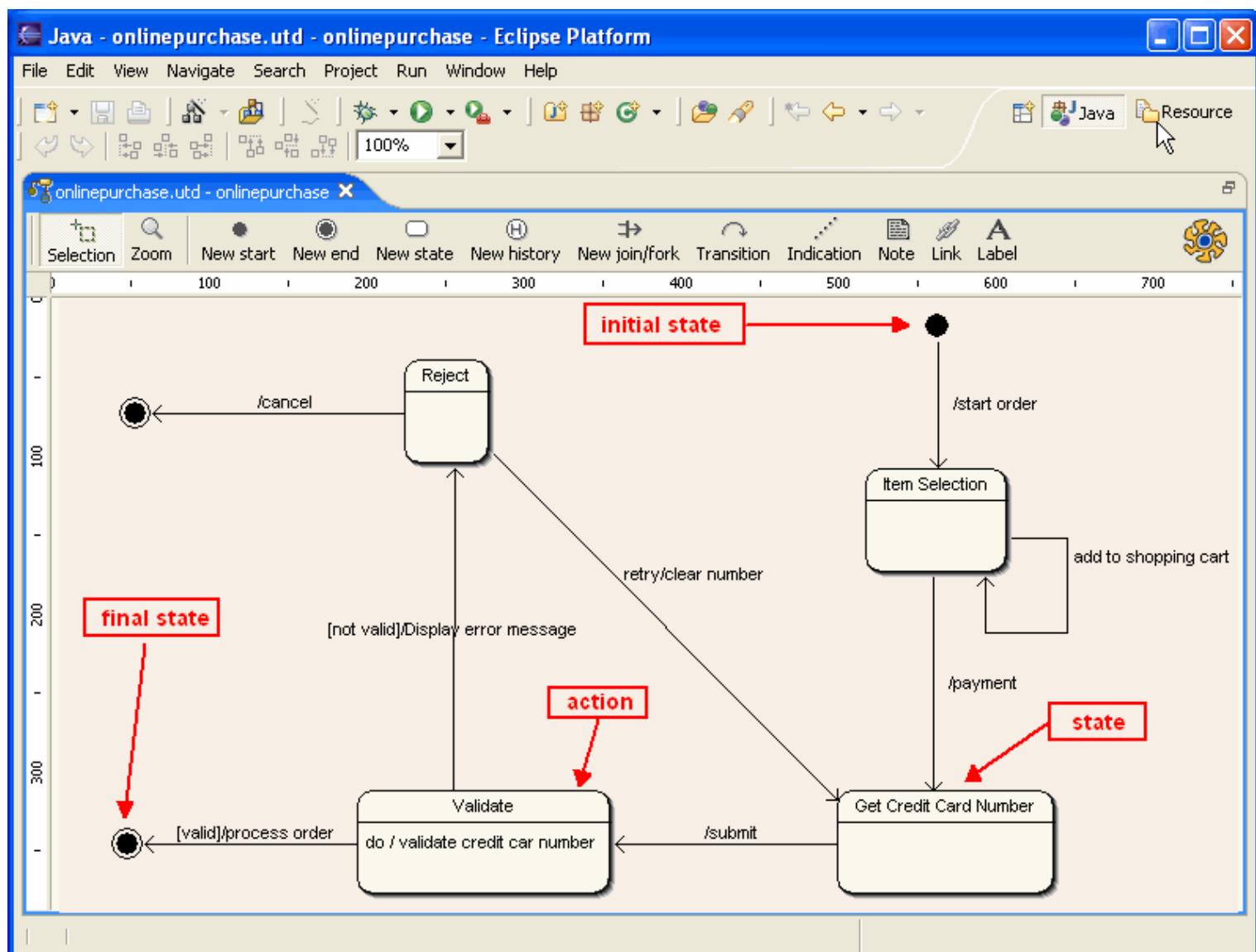
- explore the complex behavior of a class, actor, subsystem or component etc...
- model real-time systems

The elements of the State diagrams are available in the tool bar:

- An initial state item represents the first active state within a diagram.
- A final state item represents the end of the active state
- State items indicate the states of the elements modeled by the diagram
- ⌚ This tool creates a history item which is used to access the previous state of a state item. A history state item can only be created into a state item.
- ⇒ A join/fork item is used to synchronize (ie join) or fork transitions.
- ⟳ Transitions represent a change in state of the elements modeled by the diagram.

State Diagram Example

The diagram that follows models the online credit card payment. Purchasing consists of selecting an item and getting the credit card number, then submitting the information for validation. Purchasing can be factored into four states: selecting an item, getting the credit card number, validating and rejecting.



Activity Diagram

Concept

UML activity diagrams are the object-oriented equivalent of flow charts and data-flow from structured development. They describe activities and flows of data or decisions between activities. Activity diagrams and state diagrams are related. While a state diagram focuses on an object undergoing a process, an activity diagram focuses on the flow of activities involved in a single process.

Activity diagrams provide a very broad view of business processes. It can be used to break down the activities that occur within a use case. An Activity Diagram could be used for describing work flows across many use cases, analysing a use case or dealing with multi-threaded applications. Activity diagrams can be used to explore the logic of:

- a complex operation
- a single use case
- several use cases
- a business process
- complex business rules
- software processes

The elements of the Activity diagrams are available in the tool bar:

- New Activity can be created by selecting the icon in the toolbar.
- ◊ New Decision can be created by selecting the icon in the toolbar.
- New Start can be created by selecting the icon in the toolbar.
- ◎ New End can be created by selecting the icon in the toolbar.
- New State can be created by selecting the icon in the toolbar.
- Ⓜ New History can be created by selecting the icon in the toolbar.
- ‡ New join/fork can be created by selecting the icon in the toolbar.
- ⟳ A transition can be created by selecting the icon in the toolbar.
- New Object can be created by selecting the icon in the toolbar.
- ⌚ Object flow can be created by selecting the icon in the toolbar.
- .FLAG New signal receipt can be created by selecting the icon in the toolbar.
- DOC New signal sending can be created by selecting the icon in the toolbar.
- New Partition can be created by selecting the icon in the toolbar.

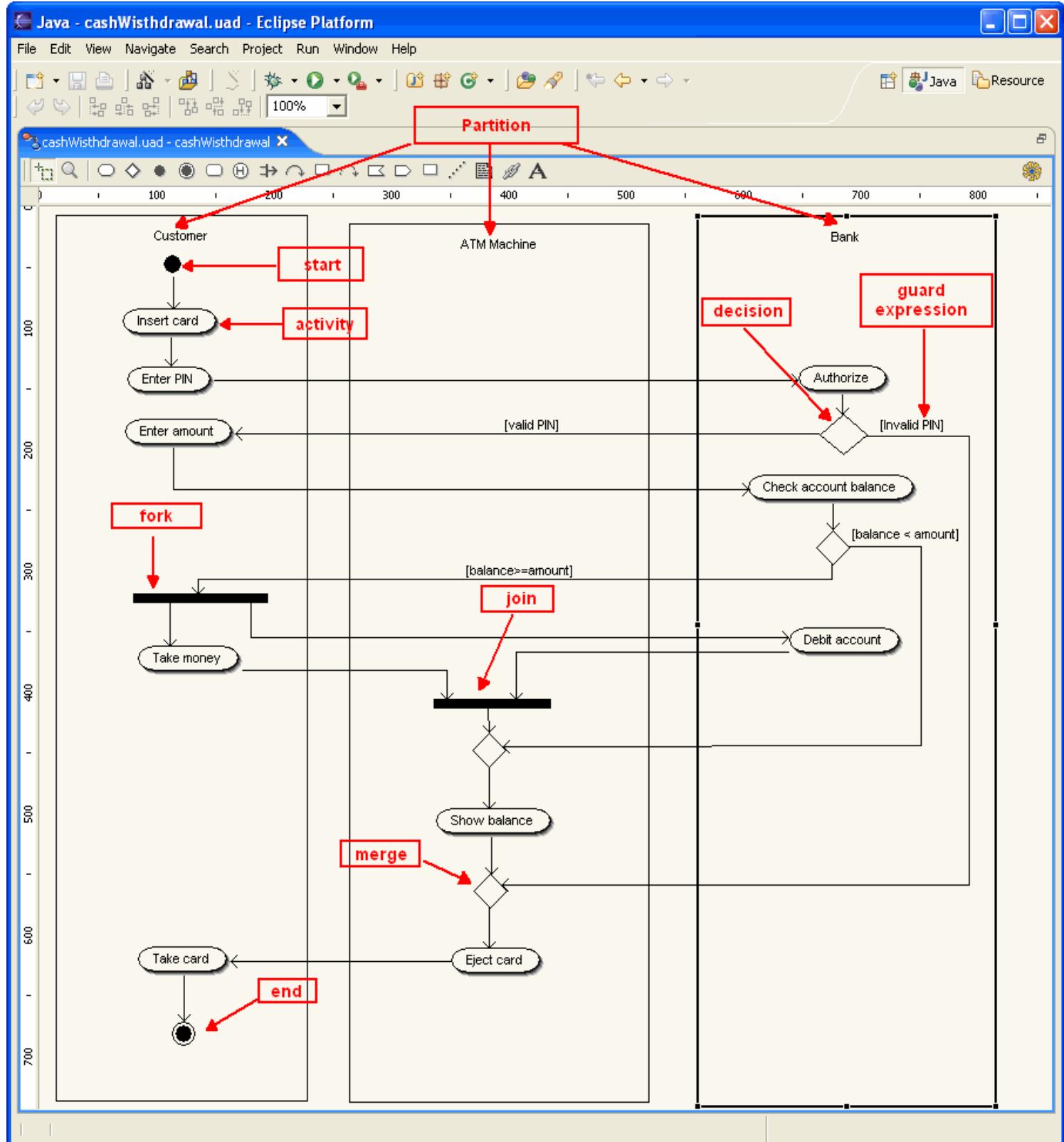
Drag and Drop

You can drag classes or interfaces from the Package Explorer to the Diagram Editor and get a new instance.

Activity Diagram Example

This diagram is useful in showing work flow connections and describing behavior that has a lot of parallel processing. When you use an activity diagram you can choose the order in which to do things.

The following example shows the withdrawal of money from an ATM Machine.
The three classes involved are Customer, ATM Machine and Bank..
Activites are shown as rounded rectangles.



The file deployment diagram extension is *.uad.

Collaboration Diagram

Concept

A Collaboration diagram is the cross between a symbol and a sequence diagram. It describes a specific scenario. Numbered arrows show the movement of messages during the course of a scenario.

Collaboration diagrams show the same information as sequence diagrams, but focus on object roles instead of the times that messages are sent.

UML collaboration diagrams, like UML sequence diagrams, are used to explore the dynamic nature of your software.

Collaborations diagrams are often used to:

- provide an overview of a collection of collaborating objects
- allocate functionality to classes
- model the logic of implementation
- examine the roles that objects take within a system

In a sequence diagram, object roles are the verticals and messages the connecting links. In a collaboration diagram, the object-role rectangles are labeled with either a class or object name.

The elements of the Collaboration diagram are available in the tool bar:

-  New Actor can be created by selecting the icon in the toolbar.
-  New Object can be created by selecting the icon in the toolbar.
-  Add new message

Drag and Drop

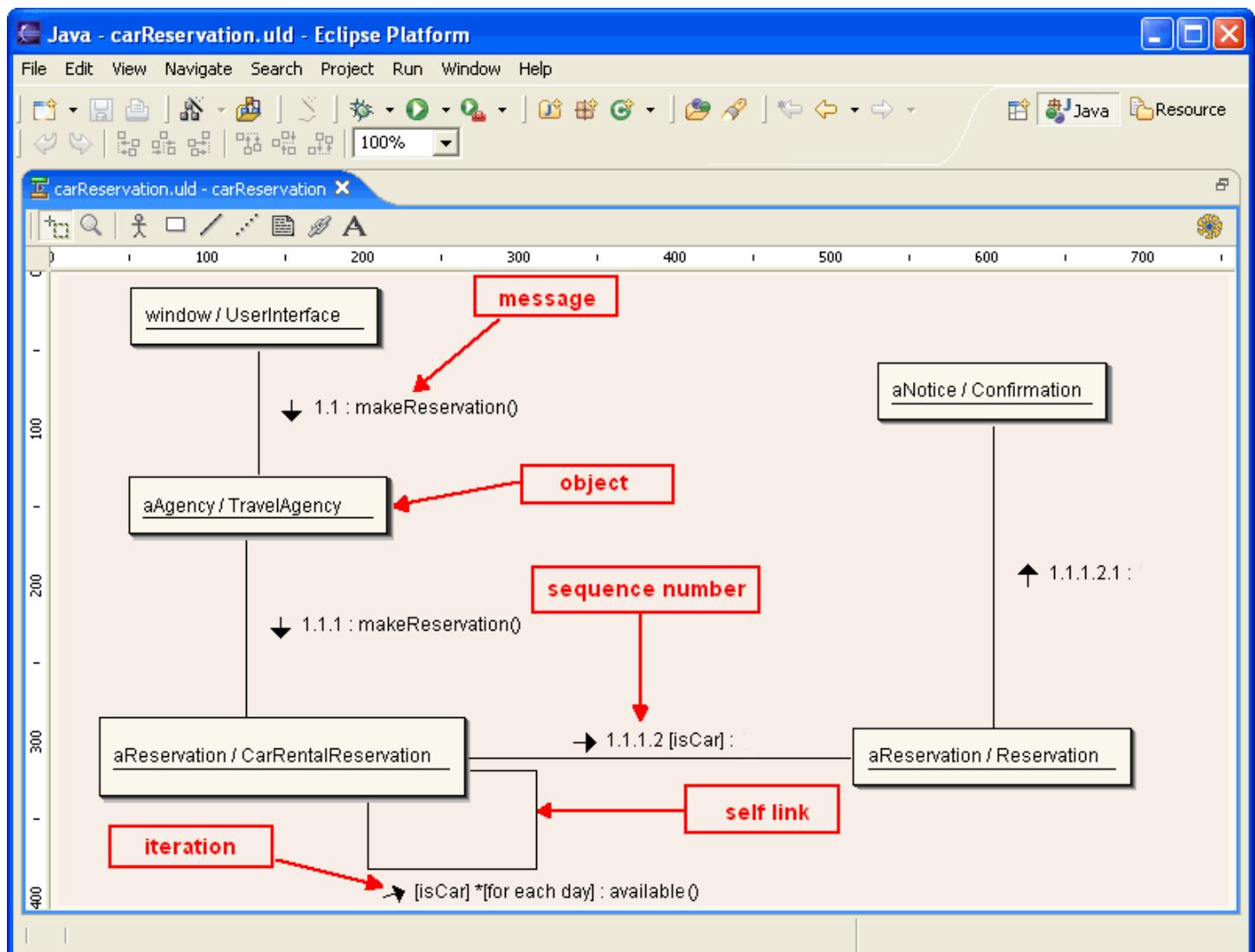
In a Collaboration diagram, you can drag and drop a class or an interface from the Package Explorer to the Collaboration Diagram Editor and get an Instance.

Collaboration Diagram Example

A collaboration diagram shows an interaction organized around the objects in the interaction and their links to each other. The following example is the same one used in the sequence diagram.

The internet browser sends a makeReservation() message to a travel agency, which sends a makeReservation() message to a car rental reservation. If the car is available, then it makes a Reservation and a Confirmation.

The car rental reservation issues a self call to determine if a car is available. If so, the car rental reservation makes a Reservation and a Confirmation.



Notes can be included.

Object Diagram

Concept

An Object diagram shows the existence of objects, their relationships in the logical view of the system and how they execute a particular scenario or use case.

Object Diagram, also called instance diagram shows an overview of a system. Using the Object diagram model, you describe the static structure of the symbols in your system.

The elements of the Object diagram are available in the tool bar:

- New Instance can be created by selecting the icon in the toolbar.
- ✍ New association between instances can be created by selecting the icon in the toolbar.
- ↳ New flow indicates how an instance is created (stereotype copy) or shows a change in state (stereotype become).

The file deployment diagram extension is *.uod.

Component Diagram

Concept

UML Component diagrams show the dependencies among software components, including the classifiers that specify them, such as implementation classes; and the artifacts that implement them, such as source-code files, binary-code files.

It is a simple, high-level diagram, which refers to physical components in a design. It is used to show how code is actually divided into modules. While package diagrams show logical or conceptual division, component diagrams are used to show physical division used for implementation.

A Component diagram shows the dependencies among software components, including source code, binary code and executable components. Some components exist at compile time, some exist at link time, and some exist at run time.

Components are physical units including:

- External libraries, COM components
- Enterprise JavaBeans and jar files
- Operating systems and virtual machines

The elements of the Component diagram are available in the tool bar:

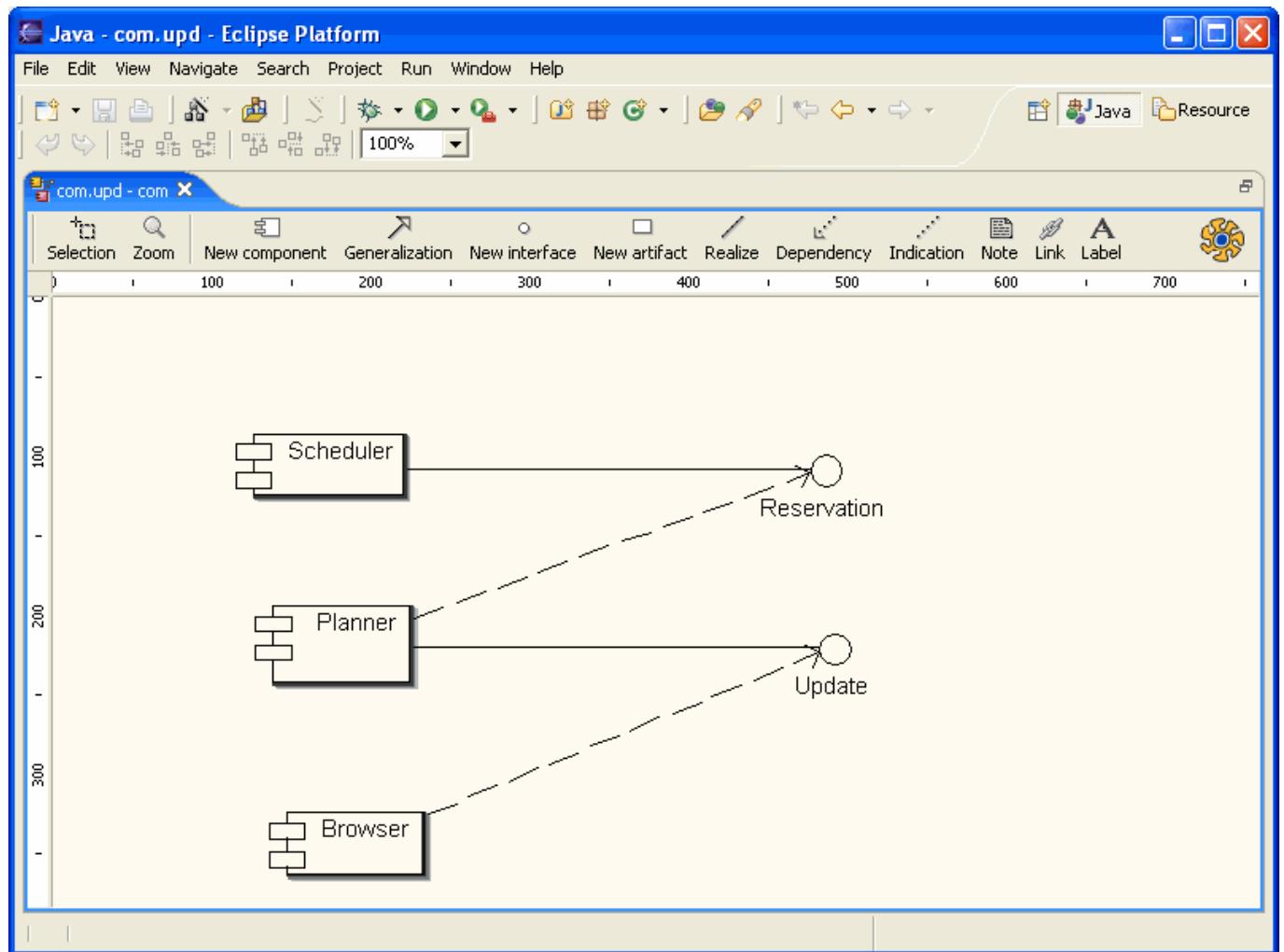
-  Create component items. Note that components can be nested.
-  Create inheritance relationships between components.
-  Add interface to the diagram.
-  Create artifact within the diagram.
-  Create an implementation relationship between a component and its interfaces.
-  Indicate dependency relationships among the diagram items.

Drag and Drop

You can drag and drop interfaces from the Package Explorer to the Diagram Editor.

Component Diagram Example

The following example is a consulting group who have consultants who use a browser to update the company schedule. The Planner then makes a new reservation in the company Schedule.



The file deployment diagram extension is *.upd.

Deployment Diagram

Concept

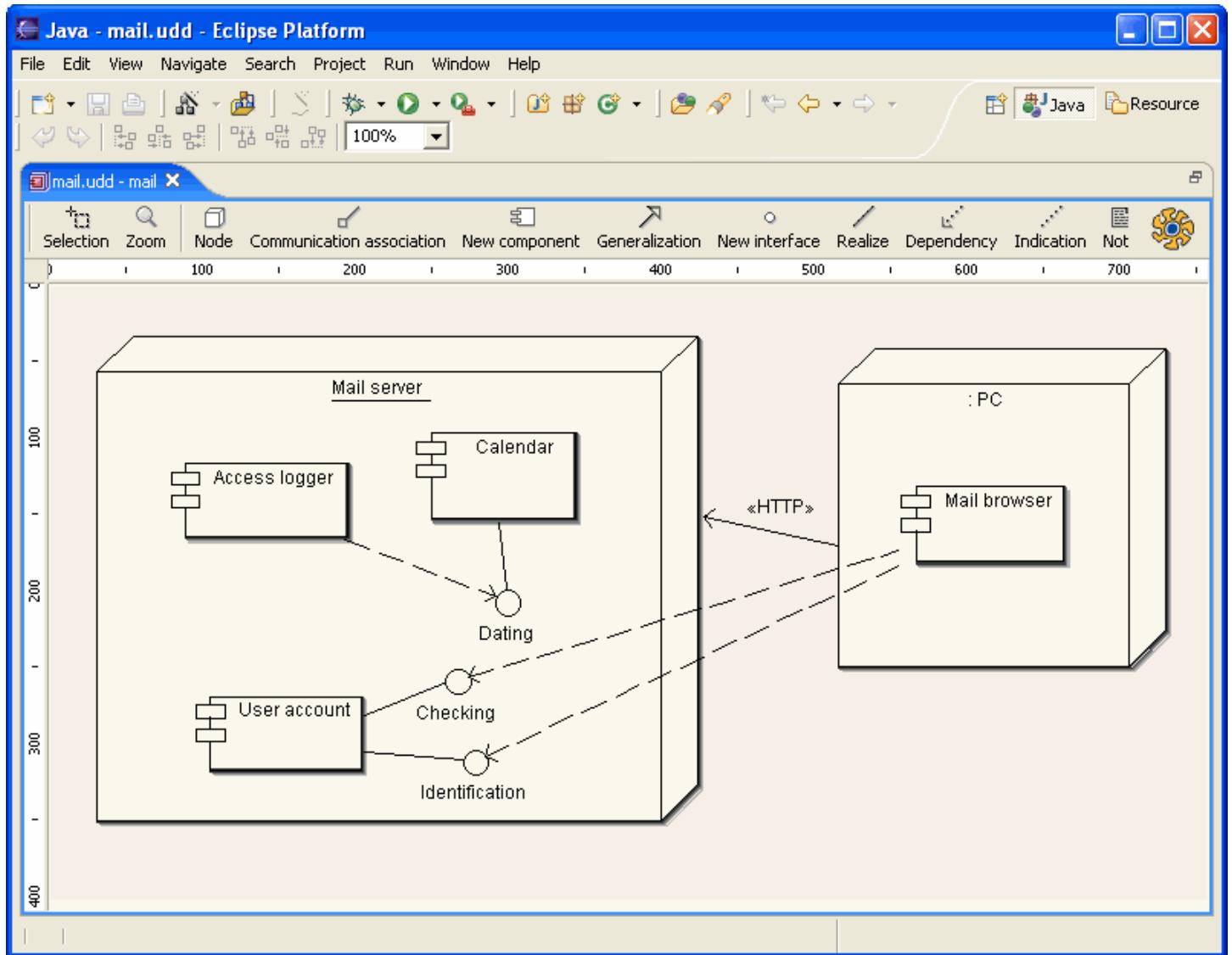
A UML Deployment diagram depicts a static view of the runtime configuration of hardware nodes and the software components that run on those nodes. It shows the configuration of run-time processing elements and the software components, processes, and objects. Software component instances represent run-time code units. Components that do not exist as run-time entities do not appear. Developers can model the physical platforms and network connections to be used in their application. Nodes usually represents hardware platforms. Component diagrams can be placed on top of deployment diagrams to indicate which modules of code will go on which hardware platform.

The elements of the deployment diagram are available in the toolbar:

-  Create node items. A node is usually used to represent a physical device.
-  Modelize communication links between nodes.
-  Create component items. Note that components can be nested.
-  Create inheritance relationships between components.
-  Add an interface to the diagram.
-  Create an implementation relationship between a component and its interfaces.
-  Indicate dependency relationships among the diagram items.

Deployment Diagram Example

The following example is a mail checking deployment diagram.



The file deployment diagram extension is *.udd.

Robustness Diagram

Concept

A Robustness Diagram is basically a simplified UML communication/collaboration diagram which uses the graphical symbols depicted below.

The elements of the Component diagram are available in the tool bar:

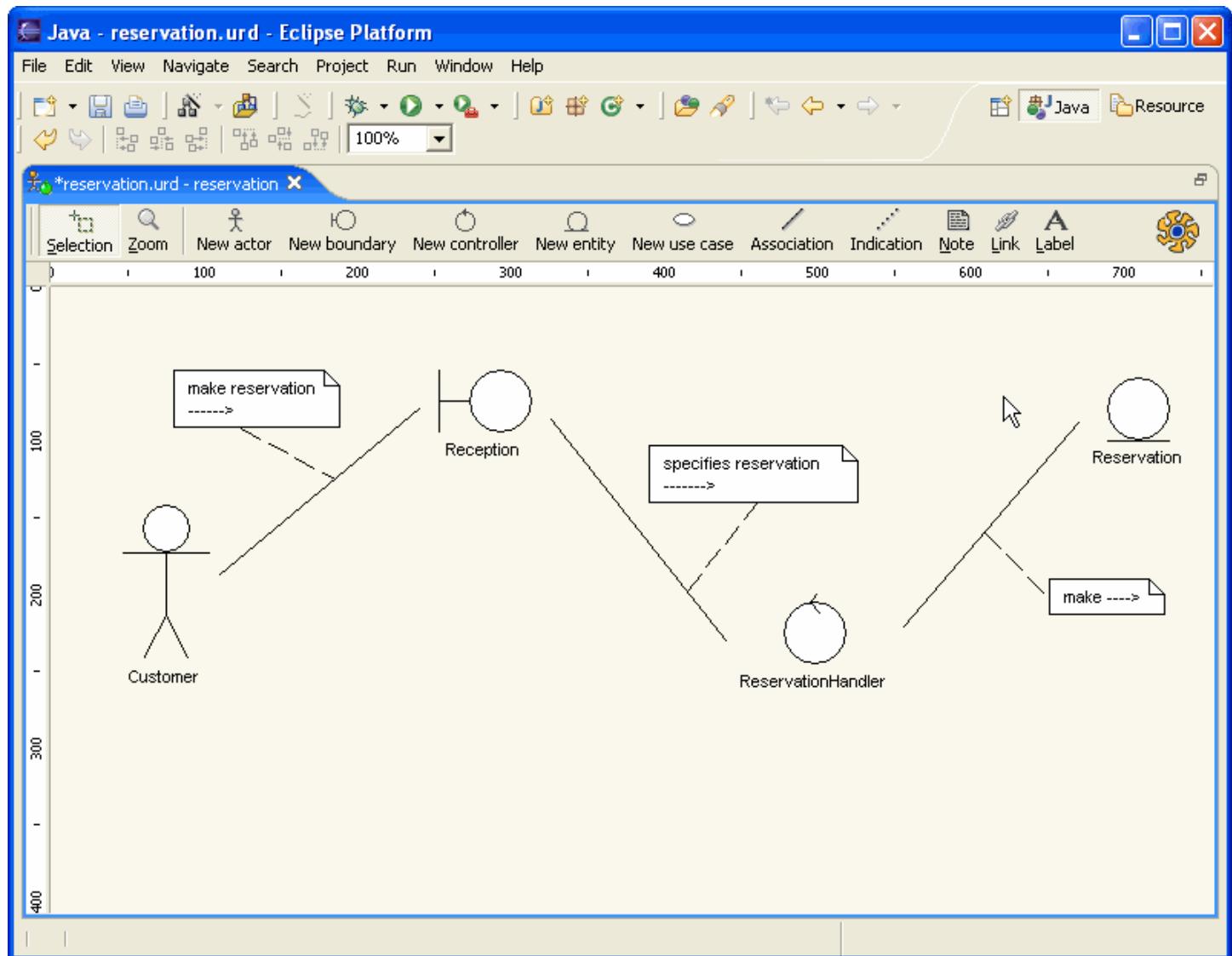
-  New actor. This is the same concept as actors in a UML use case diagram.
-  New Boundary. This represents software elements such as screens, reports, HTML pages, or system interfaces that actors interact with. Also called interface elements.
-  New controller. This serves as the glue between boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions.
-  New entity. These are entity types that are typically found in your conceptual model.
-  Association
-  New use case. Use cases can invoke other use cases you need to be able to depict this on your robustness diagrams.
-  Indication

- **Actors** can only communicate with **Boundary** classes.
- **Boundary** classes can only communicate with **Control** classes and **Actors**.
- **Entity** classes can only communicate with **Control** classes.
- **Control** classes can communicate with **Boundary** classes, other **Control** classes and with **Actors**.

Robustness Diagram Example

The following example is ReservationHandler which handles the business logic of making the reservation. Reception's job is to act as an interface to the system, providing the form for data to be input.

[See the Use Case diagram example.](#)

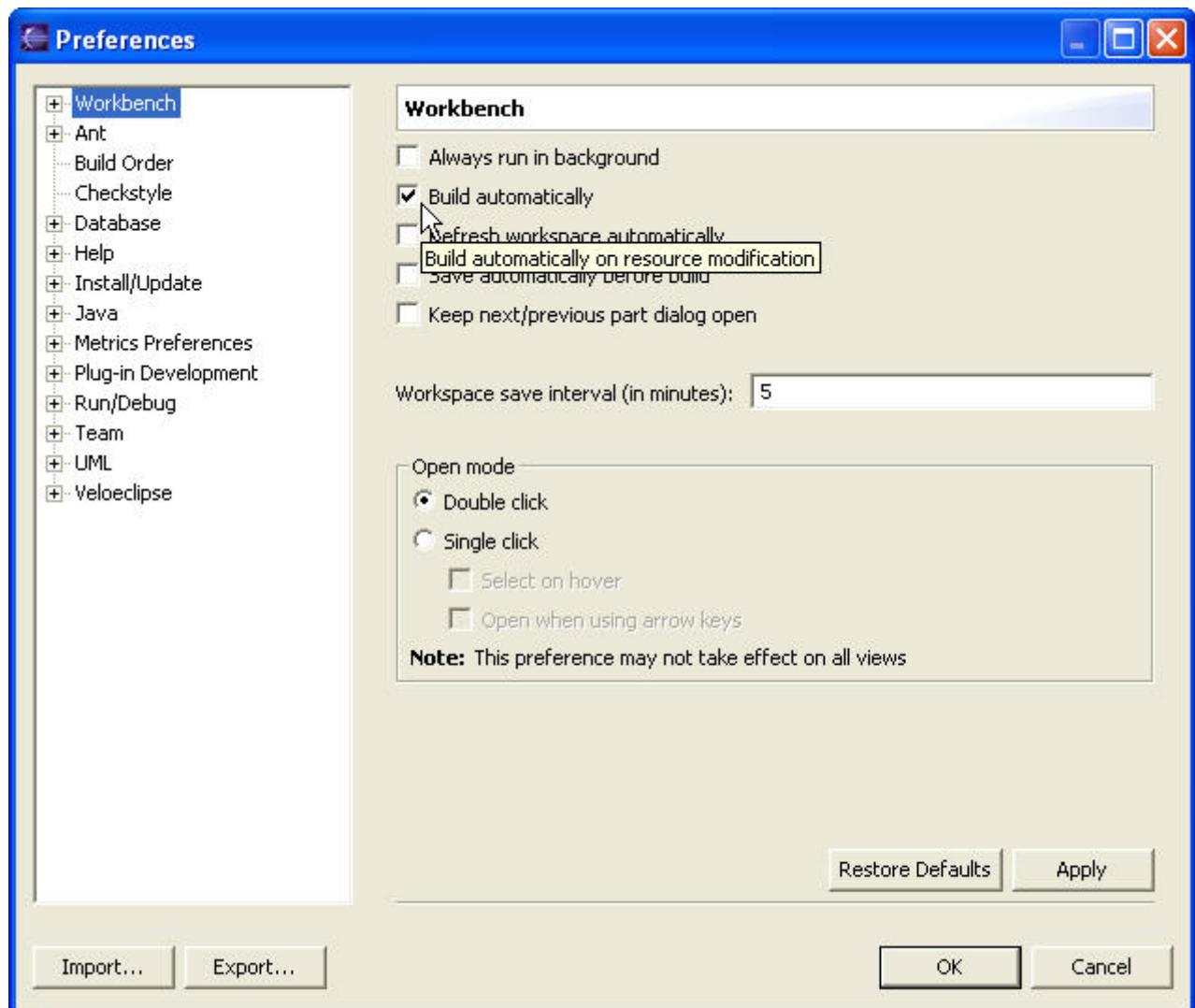


The file deployment diagram extension is *.upd.

Preferences

Workbench

The workbench build option is used by the Java Development Toolkits (JDT) to compile the Java source after each modification.



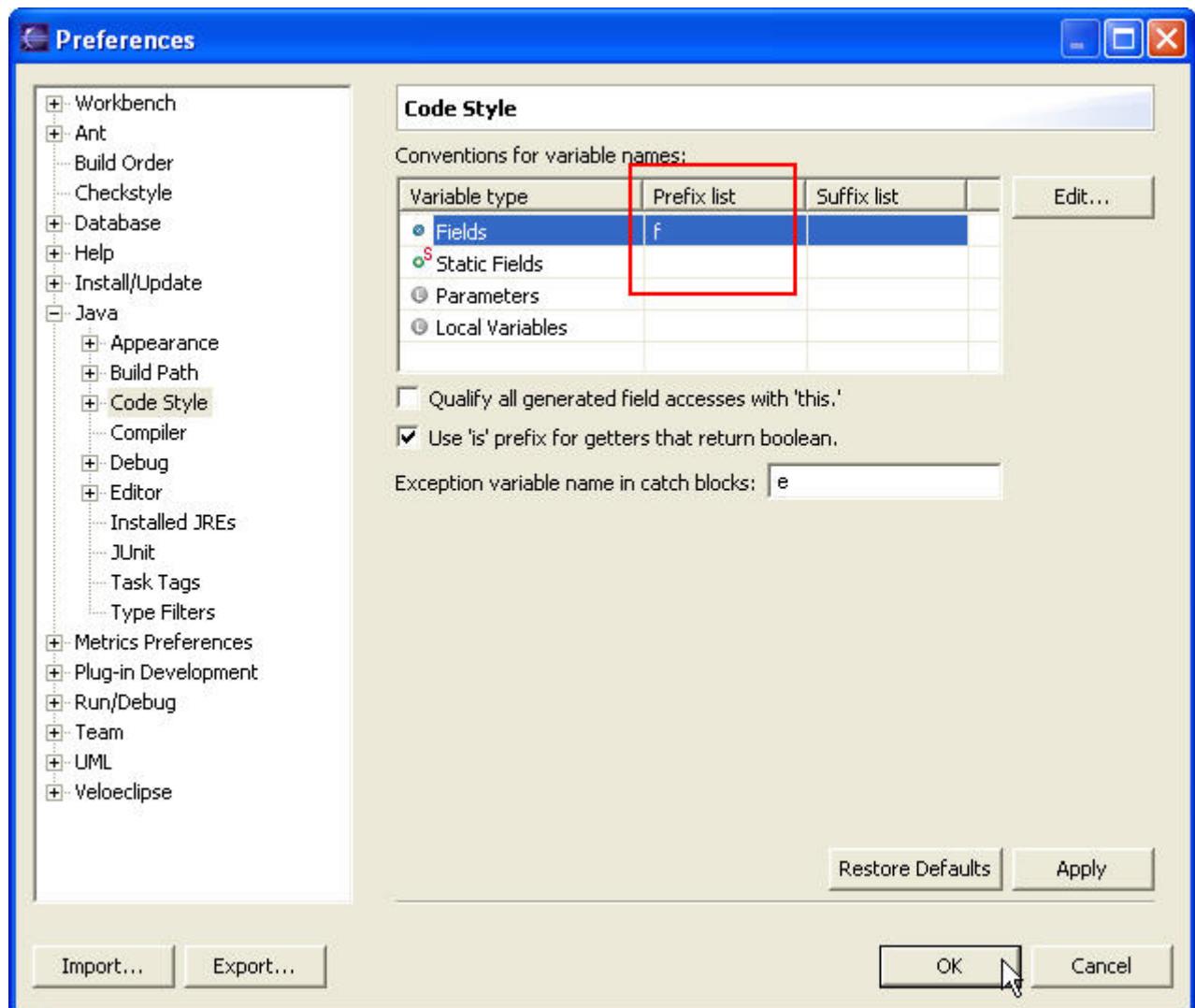
EclipseUML dependency detection and reverse engineering engine depend indirectly on this option. If this option is disable, both mechanisms may not work correctly.

JDT Preference

EclipseUML is tightly integrated in JDT. Therefore, JDT Java code generation options are used directly by two components in EclipseUML:

- [Reverse engineering](#)
- [Code generation](#)

In this example, f character will prefix any new field name.

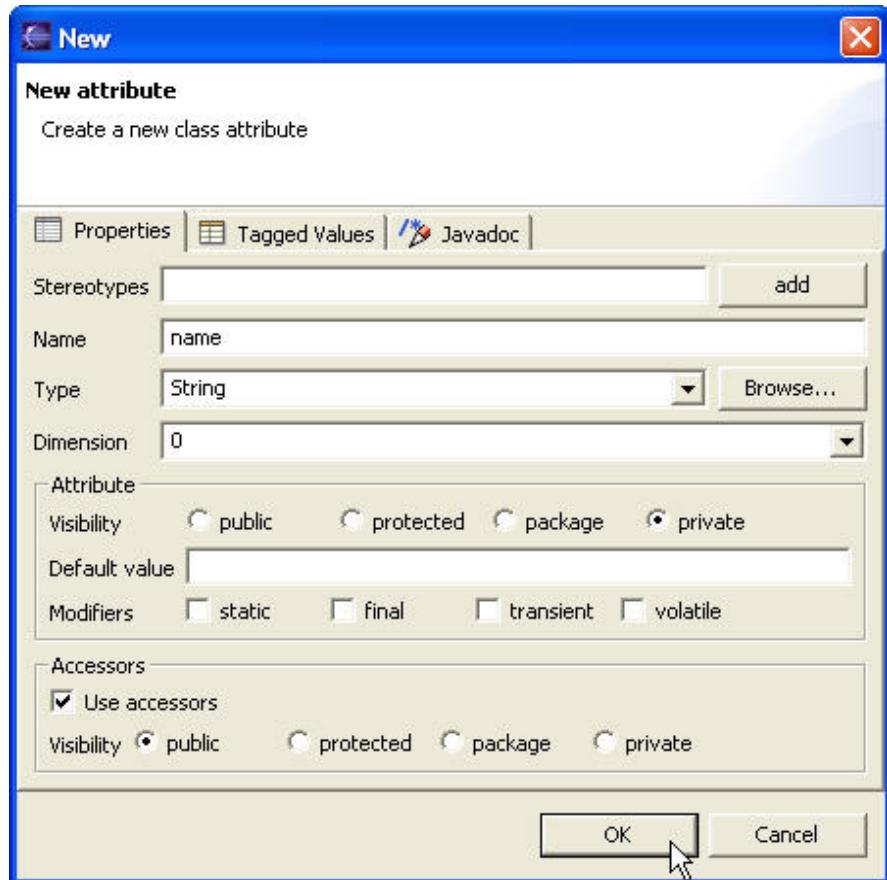


Reverse engineering

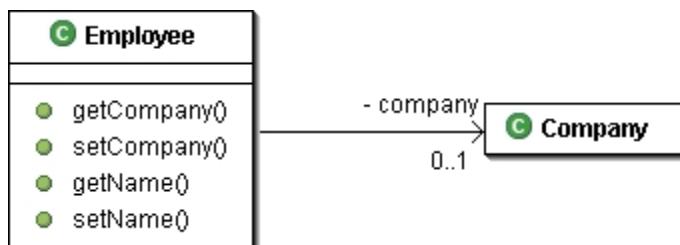
See the detail example in [Getter/Setter method recognition](#) in UML Model Reverse Engineering

Java Code generation

EclipseUML uses this option to generate Java code for attributes. When you create an attribute via diagram, EclipseUML code generator will concatenate the first prefix/suffix in this option to produce the Java attribute name. Using our previous example, if we add a new attribute String name:



we will get a Java attribute **fName** instead.



```

public class Employee
{
    /**
     * @uml.property="company" associationEnd="{}{multiplicity={(0 1)} ordering=ordered
elementType=
     * company.Company}
     */
    private Company fCompany;

    /**
     * @uml.property="company"
     */
    public Company getCompany()
    {
        return fCompany;
    }
}
  
```

```

/**
 * @uml property=company
 */
public void setCompany(Company company)
{
    fCompany = company;
}

/**
 *
 * @uml property=name
 */
private String fName;

/**
 *
 * @uml property=name
 */
public String getName()
{
    return fName;
}

/**
 *
 * @uml property=name
 */
public void setName(String fName)
{
    this.fName = fName;
}
}

```

Setting Preferences

The Omondo mechanism of preferences is working at three different levels: global/diagram/element.

1. Eclipse preferences:
This includes workbench settings and JDT settings.
(first level)
2. Diagram preferences:
(second level)
3. Element Preferences:
(third level)

Depending on its nature, an element can keep its preferences.

Every element of a diagram, or diagrams keep their own preferences.

Changing global preferences will not impact your existing diagram's or element's preferences.

Different preferences can be introduced using:

1. The OK button will only change new diagrams or elements and will not change your existing diagrams.
2. The Apply button will dynamically change all existing diagrams' preferences.
3. Restore Default will automatically select the preference of the higher level. For example, clicking on Restore Default in the element preferences (third level), will automatically select the diagram preferences (second level). Clicking on Restore Default in the diagram preferences will select global preferences (first level).

Global EclipseUML Preferences

The global Eclipse preferences are used for every project. It is the highest level of preferences you can specify.

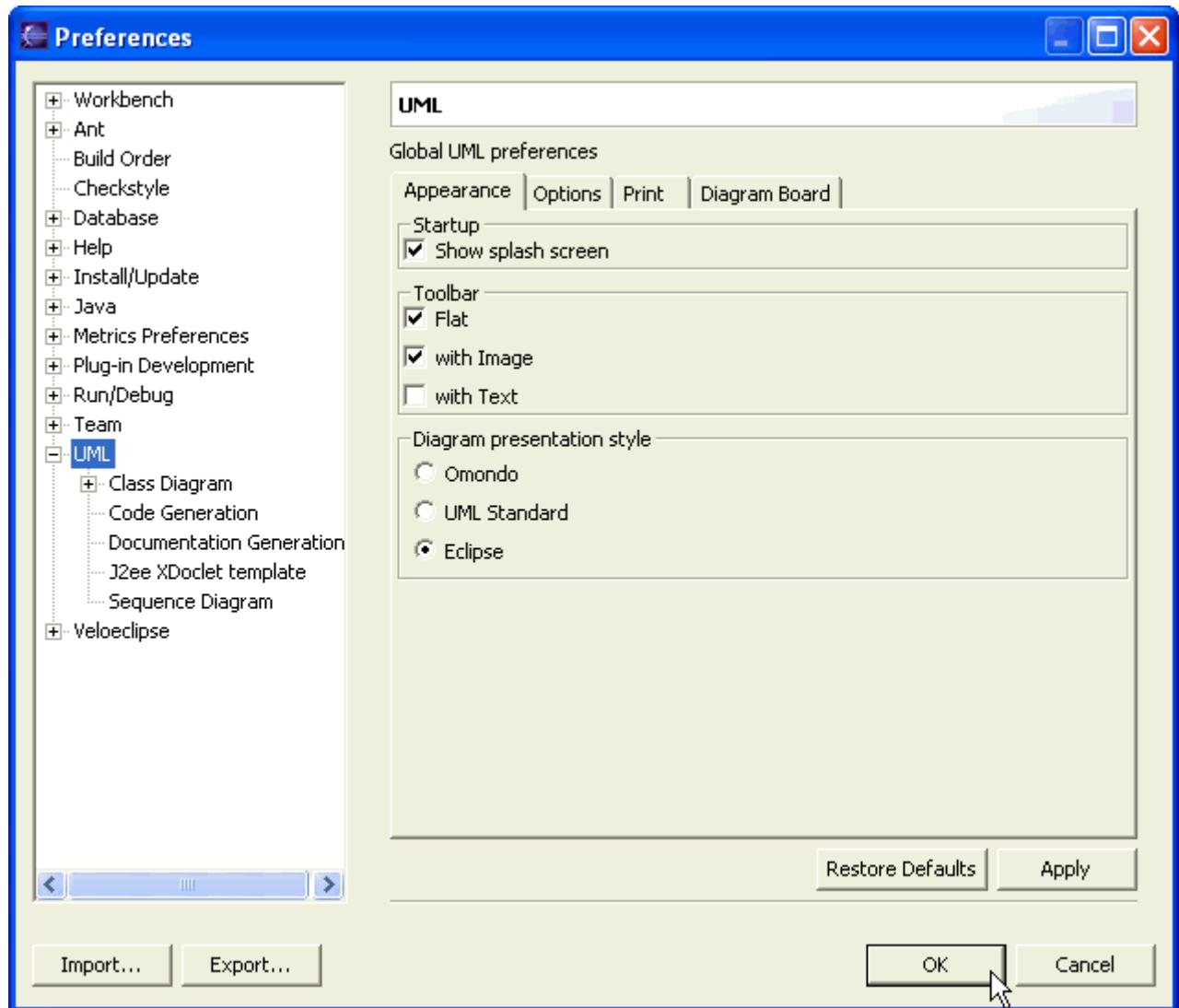
Three options are available:

1. [Appearance](#)
2. [Option](#)
3. [Print](#)
4. [Diagram Board](#)

1. Appearance

Appearance preferences represent the general EclipseUML presentation when you first start modeling. Three options are available:

- Startup
- Toolbar
- Diagram presentation style



Startup

This preference allows you to active the Omondo splash screen or not.

Toolbar

This preference allows you to choose the toolbar appearance.

Multiple possibilities are provided :

- Flat (the toolbar buttons are flat, with no relief)
- with Image (icons are available)
- with Text (text is printed)

By default, *flat* and *with Image* are selected, text is only visible on tooltips.

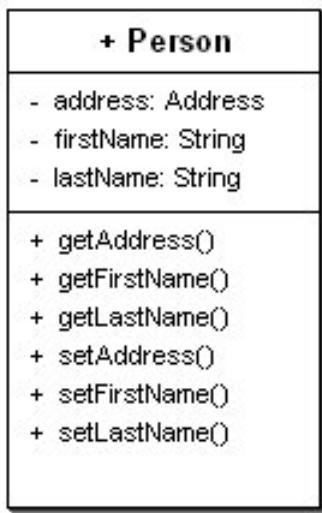
Diagram presentation style

This property allows you to choose a style for your UML diagram's appearance. Three possibilities are available :

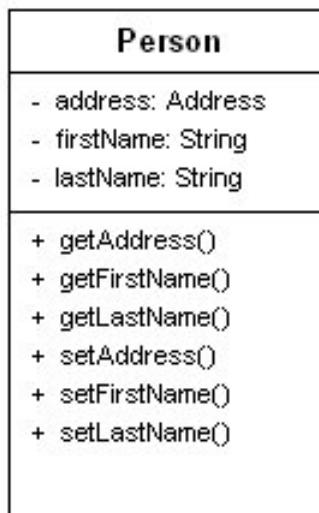
- Omondo (this look and feel, designed by Omondo, will evolve)
- UML Standard (the standard look and feel, just as usually in UML diagrams)
- Eclipse (The Eclipse look and feel, using the same colors and icons)

By default, the presentation style is set to *Eclipse*.

Here is an example of each style on a single class :



Omondo



Standard



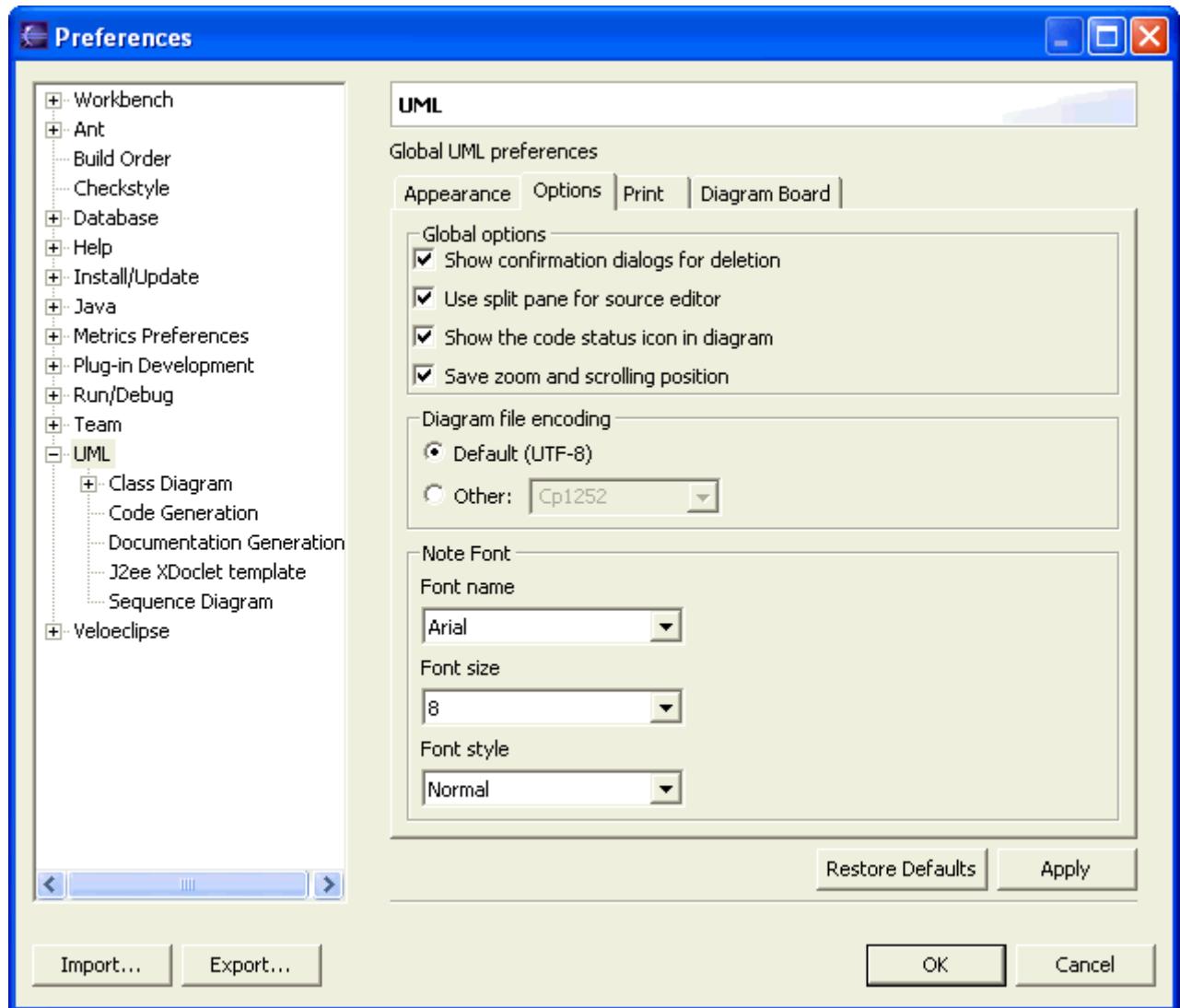
Eclipse

2. Options

Options Preferences represent the general EclipseUML customization.

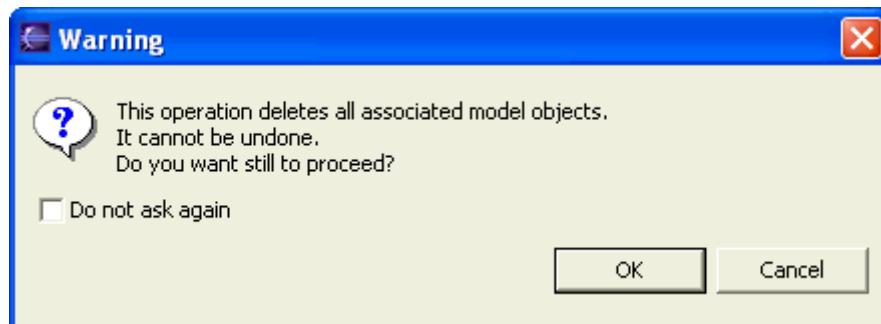
Four options are available:

- Global options
- Diagram file encoding
- Note Font



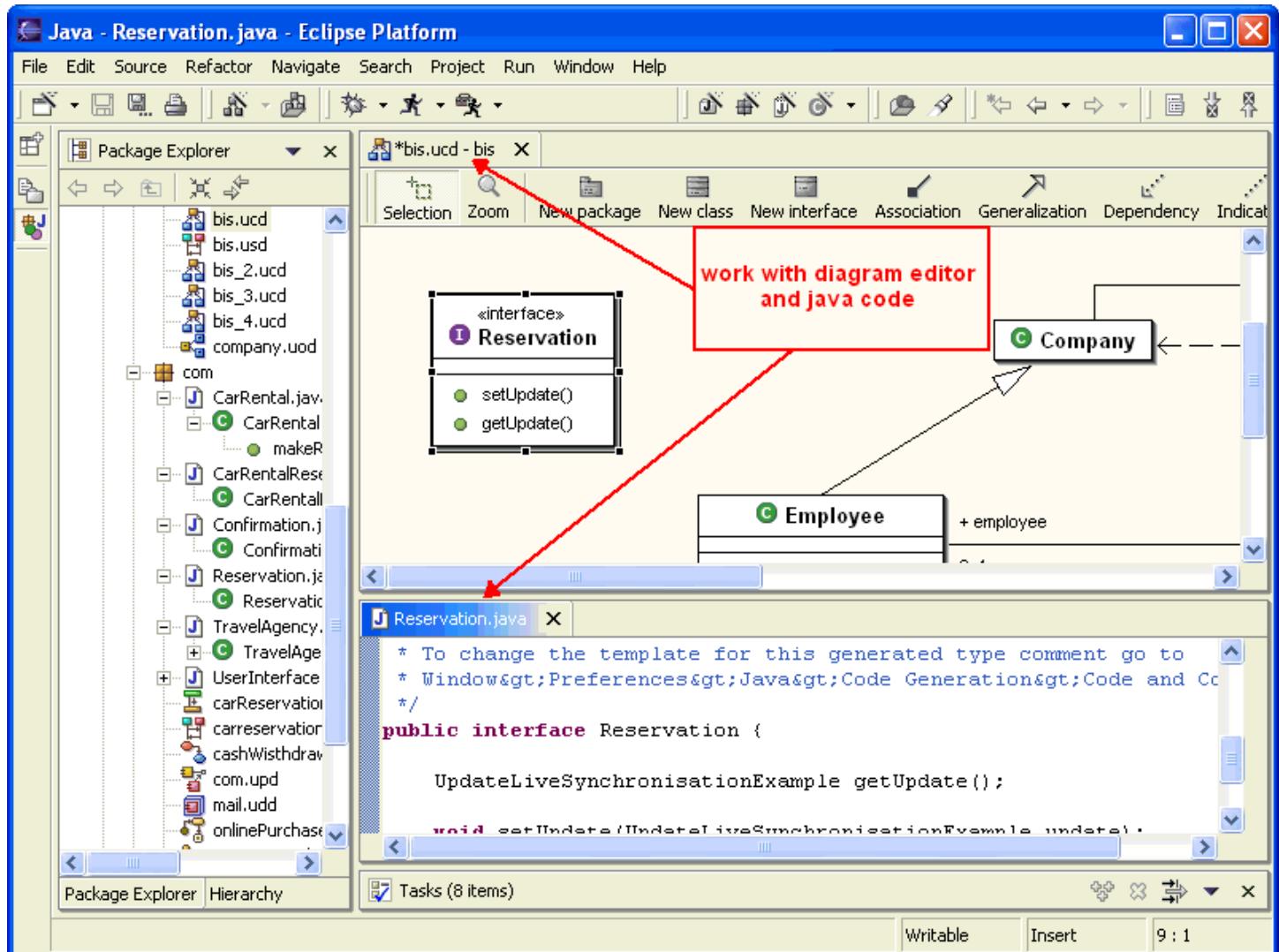
Global options

Show confirmation dialogs for deletion allows you to delete any element, with or without any warning. If you select this option then the warning will not appear when deleting an element. In order to reactivate this warning window, you must select the global preferences and unselect "Do not ask again".

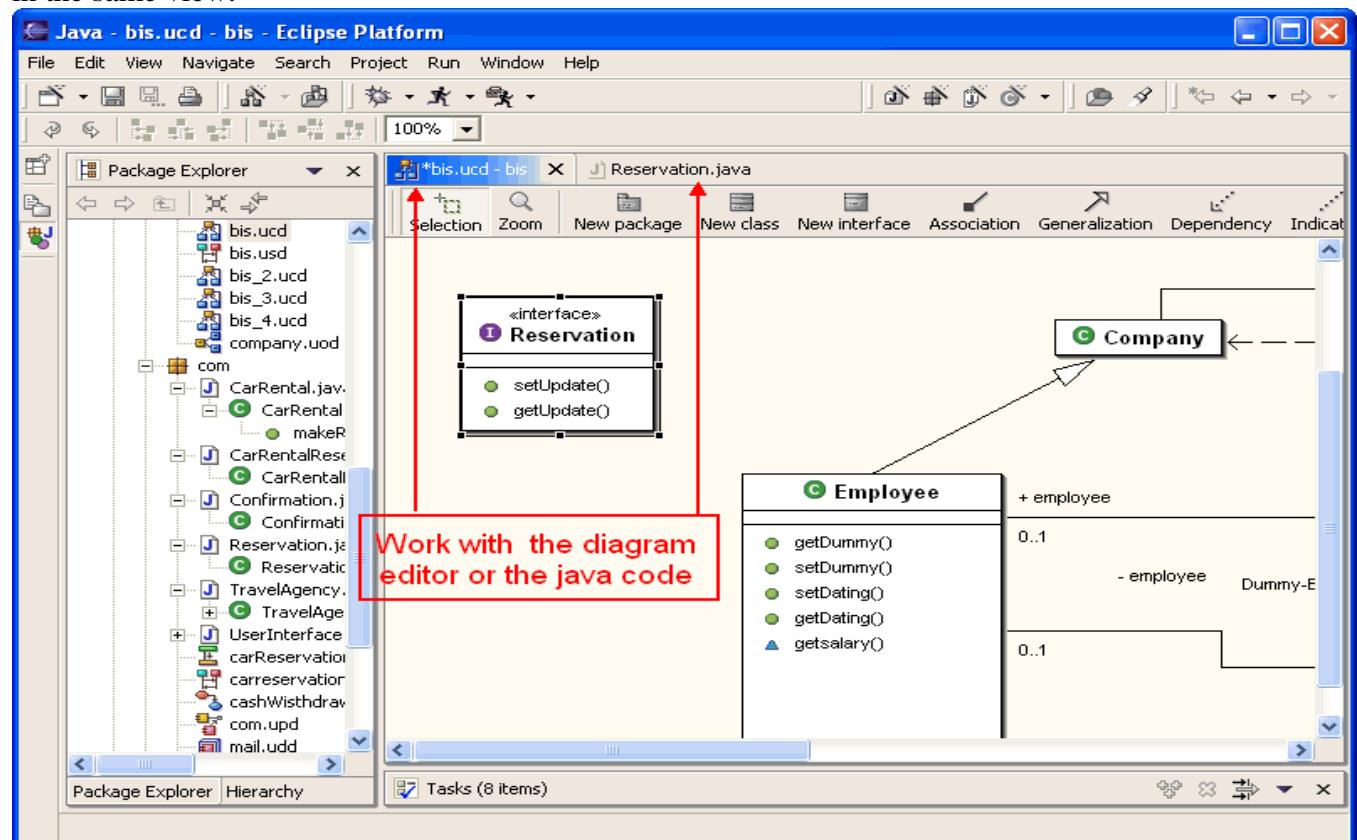


Use split pane for source editor allows you to have two different and simultaneous views of each editor or to keep both diagram and source editors in the same view.

If you select this option, you will be able to work with the diagram editor and the Java code in the same view.

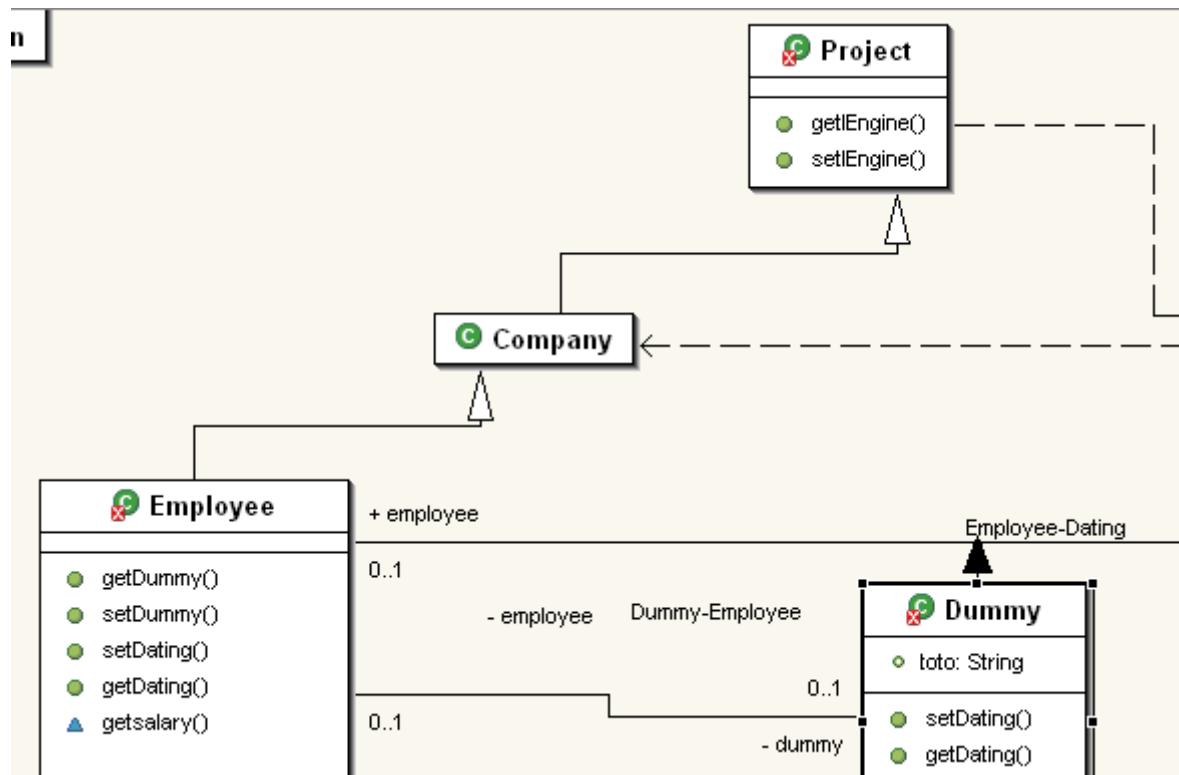


If you do not select this option, you will not be able to work with the diagram editor and the Java code in the same view.

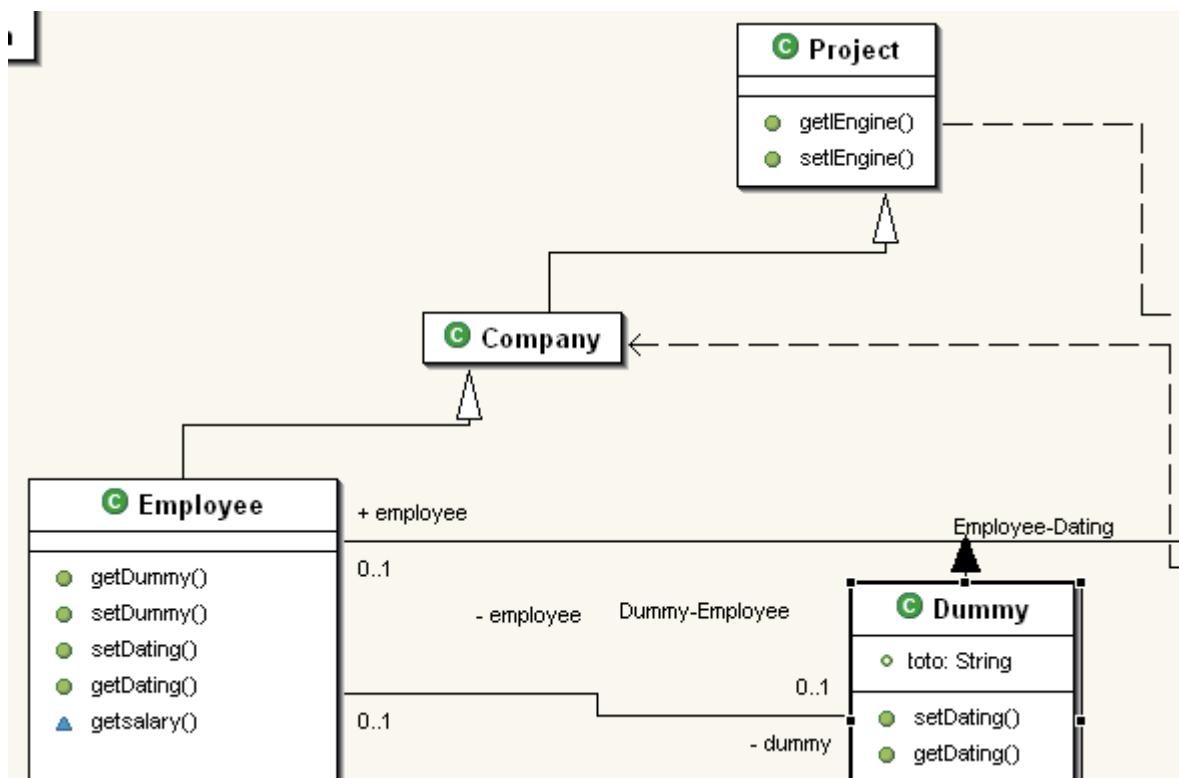


Show the code status icon in diagram is useful if using Eclipse diagram presentation style.

If you select show the code status, every error will be visible in the diagram editor:



If you do not select show the code status, errors will not be visible in the diagram editor:



Save zoom and scrolling position allows you to reopen a diagram as it was before it was closed. This is important in order to start your work exactly where you finished it before.

Encoding

The defined encoding is used when diagrams are saved.

Default (UFT-8) is recommended.

When using special characters in diagrams (including notes), it is important to save the diagram with an appropriate encoding. This is especially true when using Japanese, Chinese, or Korean languages.

Note Font

An appropriate font must be selected for exporting images including special characters.

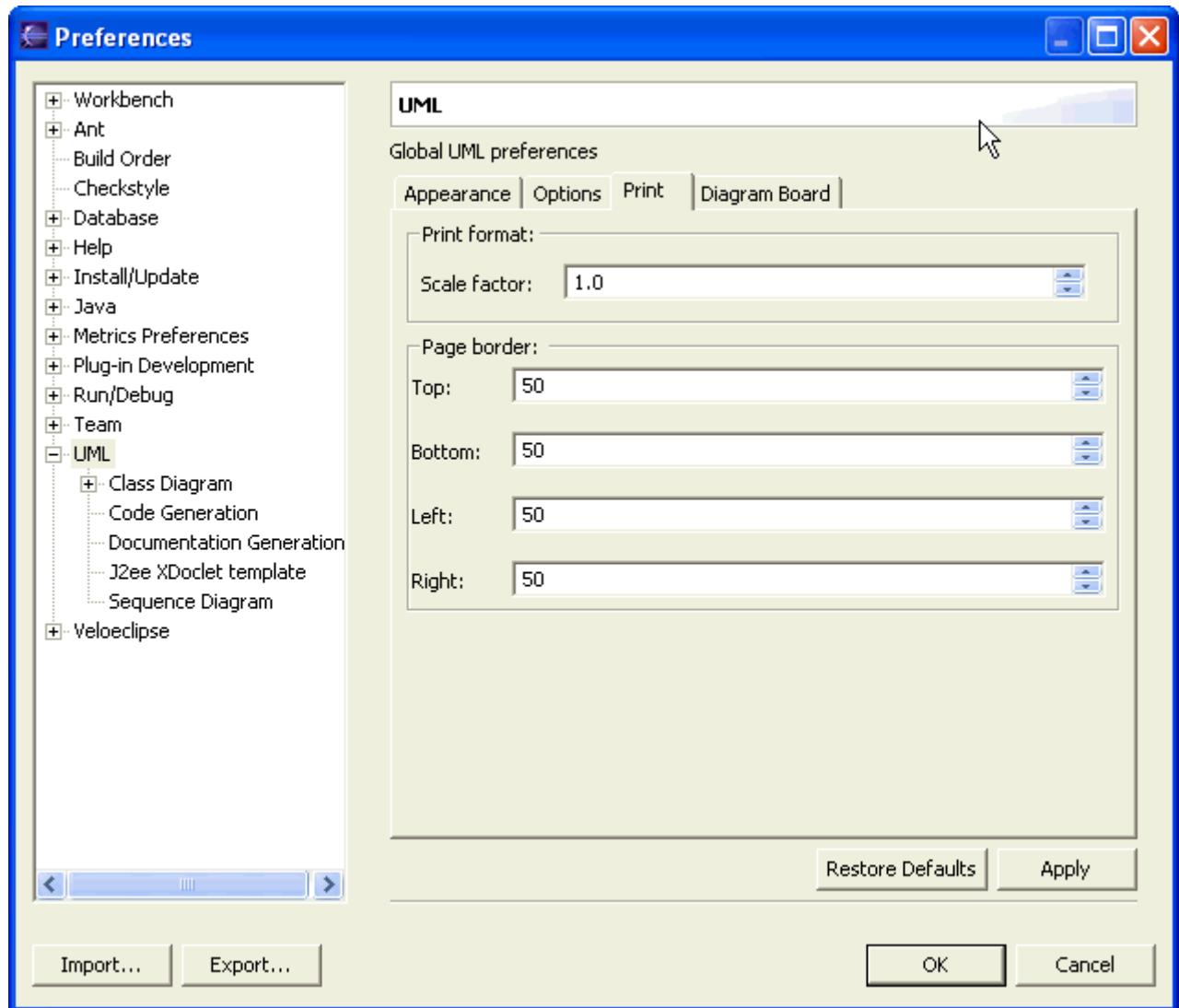
This is especially true when using Japanese, Chinese or Korean languages as the default font (Arial 12) doesn't support such characters.

3. Print

Options are proposed to configure the [diagram printing](#).

Two options are available:

- Print format
- Page Border



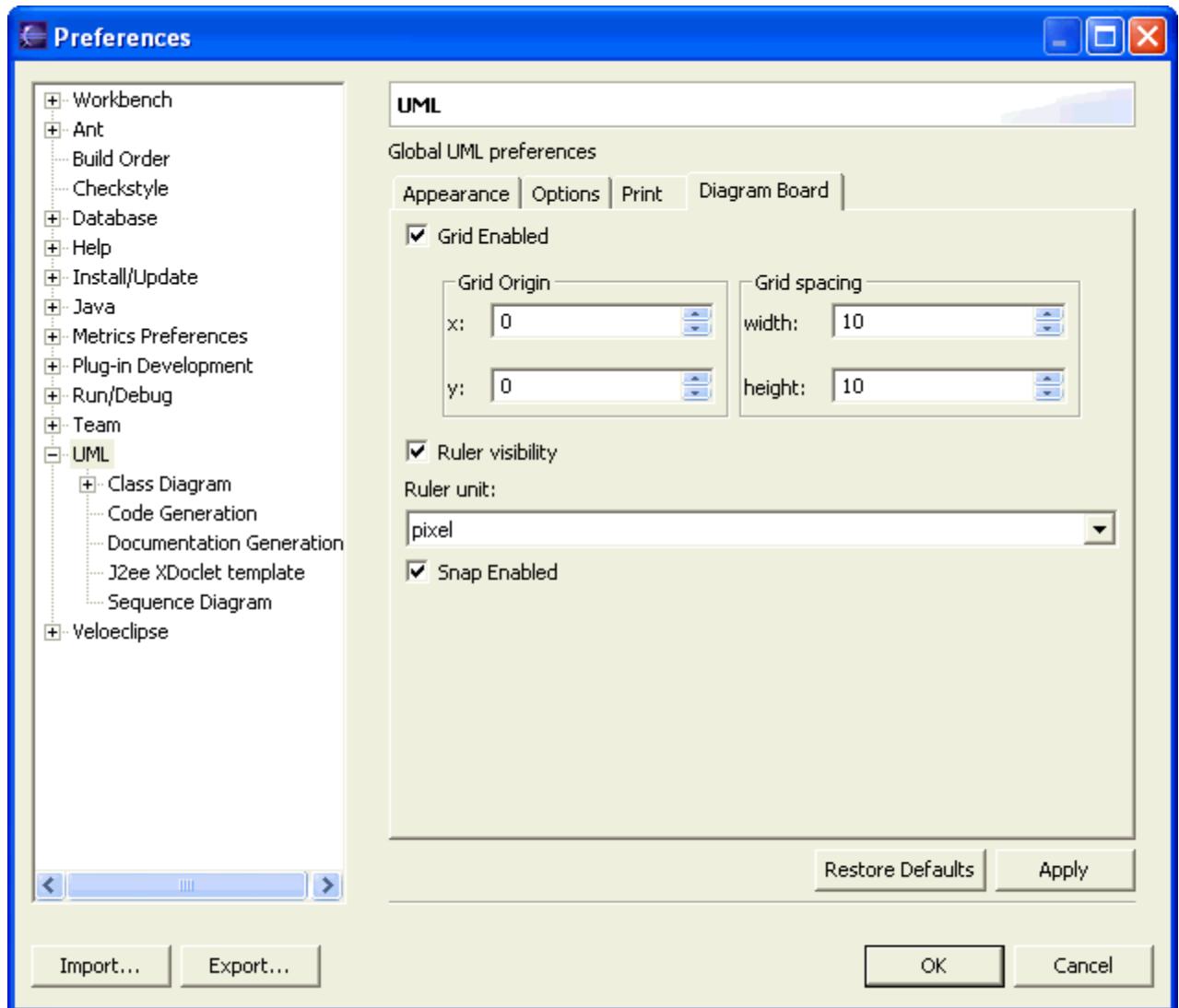
Print format represents the scale factor. The scale factor 1.0 is equivalent to a 100% presentation and 2.0 is equivalent to 200%. It is useful to visualize the overview before printing.

Page border represents the border customization of your diagram editor image and its export to your printer.

4. Diagram Board

Options are proposed to configure the diagram grid and ruler. Six options are available:

- Grid Enabled
- Grid Origin
- Grid spacing
- Ruler visibility
- Ruler unit
- Snap Enabled



Class Diagram Preferences

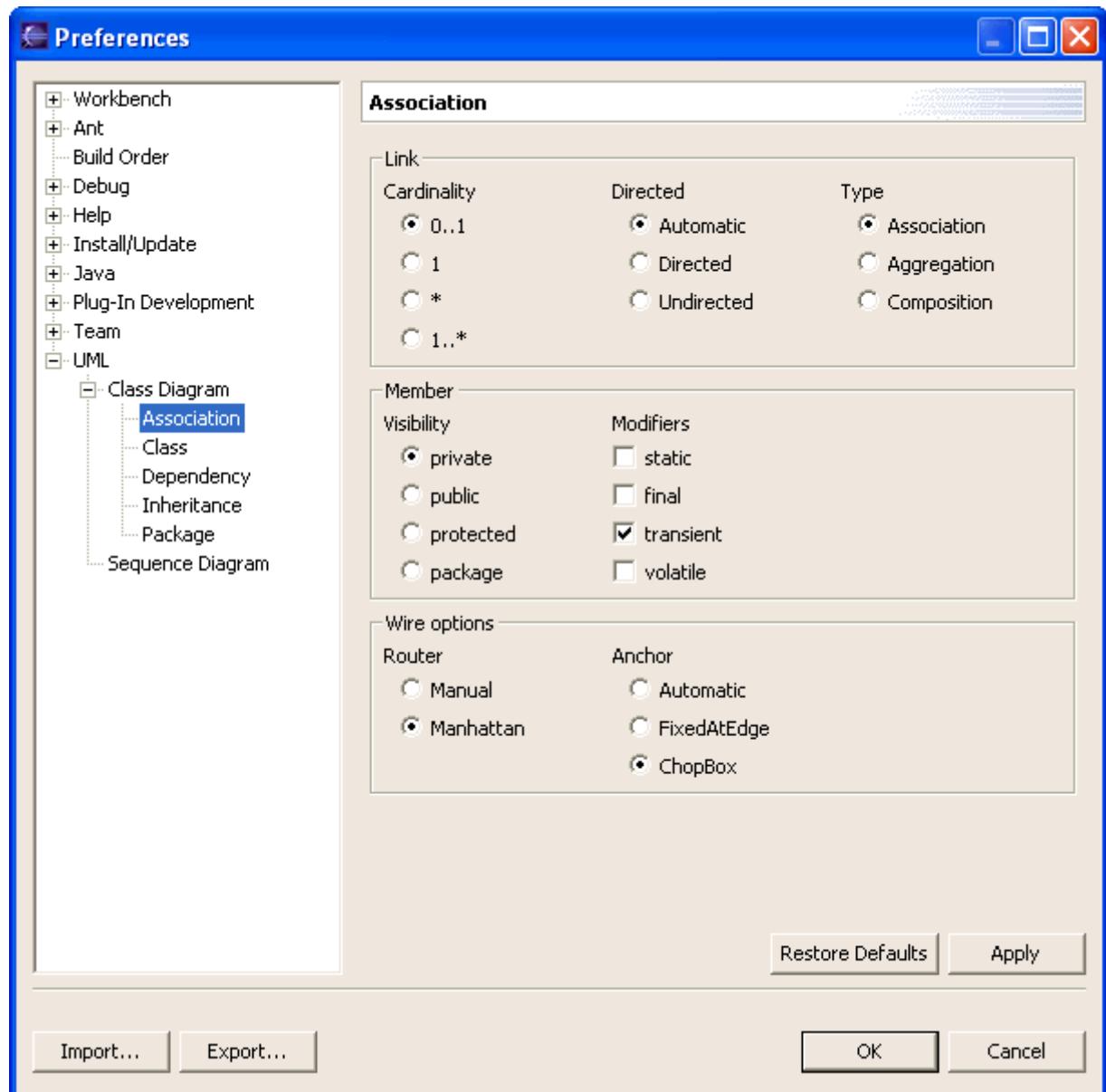
The UML preferences can be selected from the menu bar: **Window > Preferences > UML > Class Diagram**.

The class diagram preferences are set in the UML preferences submenu.

- [Association](#)
- [Class](#)
- [Colors](#)
- [Dependency](#)
- [Inheritance](#)
- [Package](#)

Associations

Associations can be customized from the preferences (UML > Class Diagram > Association).

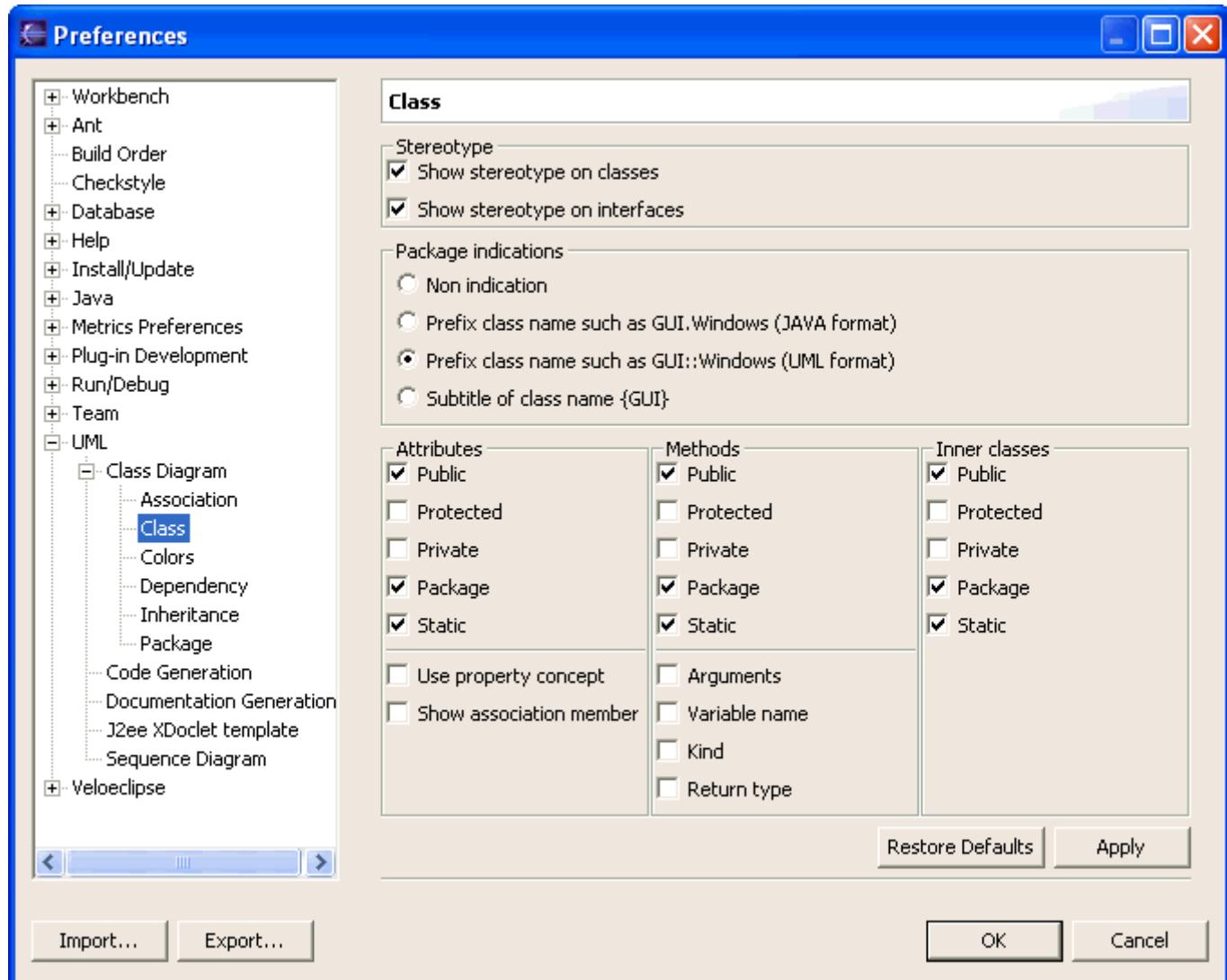


Three options are available:

- 1. From the Link area, the cardinality, the direction and the type of the association can be defined.
 - Cardinality
 - Direction
 - Type
- 2. From the member area, the visibility and the modifiers of the new members created as association ends can be defined.
- 3. From the wire area, [the router and the anchor](#) of the association can be defined.
 - The router link can be manual, or designed automatically using the Manhattan option
 - Anchor option can be automatically fixed with Chopbox or manually fixed using FixedAtEdge.

Class Preferences

Classes and Interfaces can be customized from the preferences (UML > Class Diagram > Class).



Five major options are available:

1. Stereotype

This preference allows you to choose:

- Show stereotype on classes
- Show stereotype on interfaces

2. Package indications

Usually, classes and interfaces of a class diagram come from the same package. But sometimes they can come from different packages, and indication about the package from which classes or interfaces come from might be useful. This preference allows you to set how package information should be shown in the class diagram. This only applies to new classes or interfaces, any modification will not impact existing classes.

Four ways of package indications are available :

- No indication (only the class name is displayed)
- In a package view
- Prefix class name such as GUI:Windows (Classes from a different package are prefixed with their package name)
- Subtitle of class name {GUI} (Classes from a different package are subtitled with their package name)

3. Attributes

New functions:

Use property Concept: By default, the [property concept](#) is activated, so the attribute and its accessors are displayed as a single property. ([See getting started example.](#))

Show association member: By default, the show association member is not activated, ([See getting started example.](#))

This preference allows you to choose which attributes will be visible in your UML view. It is only applied to new classes, any modification will not impact existing classes.

Multiple possibilities are provided :

- Public (The public attributes will be visible).
- Protected (The protected attributes will be visible)
- Private (The private attributes will be visible).
- Package (The package attributes will be visible. An attribute without visibility modifier has the package visibility).
- Static (The static attributes will be visible if their visibilities are visible).
- Use property concept (The attributes and their accessors will be displayed as properties)
- Show association member (The attributes which are association ends will be displayed)

More detail is then provided for each class in the diagram through the [view selector](#).

4. Methods

This preference allows you to choose which methods will be visible in your UML view. It is only applied to new classes, any modification will not impact existing classes.

Multiple possibilities are provided :

- Public (The public methods will be visible).
- Protected (The protected methods will be visible)
- Private (The private methods will be visible).
- Package (The package methods will be visible. A method without visibility modifier has the package visibility).
- Static (The static methods will be visible if their visibilities are visible).
- Arguments (The argument types are visible).
- Variable name (The argument names are visible).
- Kind (The argument kinds are visible).
- Return type (The return type is visible).

More detail is then provided for each class in the diagram through the [view selector](#)

5. Inner classes

This preference allows you to choose which inner classes will be visible in your UML view. It is only applied to new classes, any modification will not impact existing classes.

Multiple possibilities are provided :

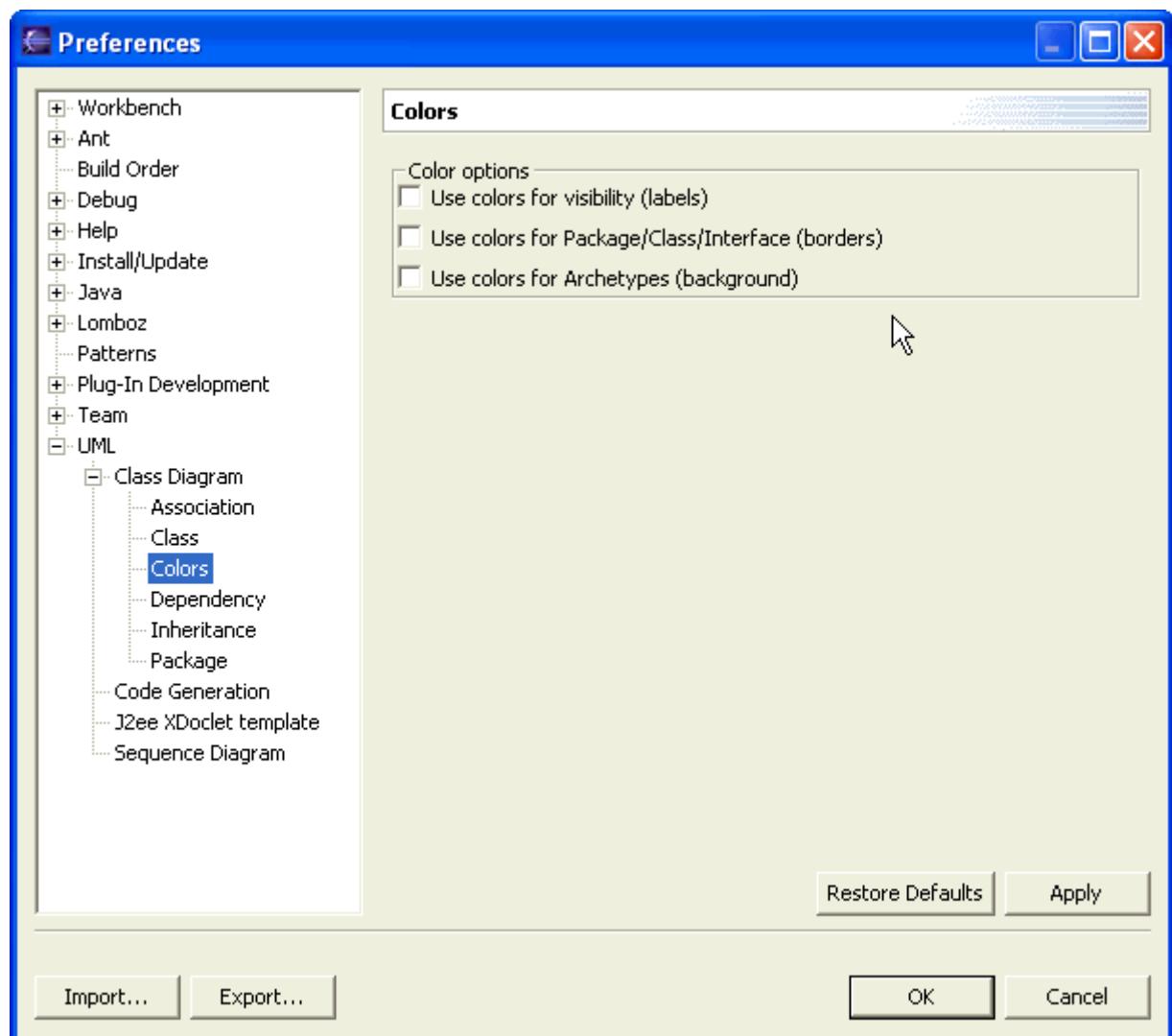
- Public (The public inner classes will be visible).
- Protected (The protected inner classes will be visible)
- Private (The private inner classes will be visible).
- Package (The package inner classes will be visible. An inner class without visibility modifier has the package visibility).
- Static (The static inner classes will be visible if their visibilities are visible).

More detail is then provided for each class in the diagram through the [view selector](#).

Colors Preferences

Colors can be customized inside your class diagram.

From the menu bar select **Window > Preferences > UML > Class Diagram > Colors**.



Three options are available:

1. Use colors for visibility

This preference allows you to choose the color of the text inside your diagram.

Eclipse presentation.



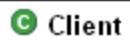
Eclipse presentation if Use of colors for visibility(labels) is selected in the menu.



2. Use colors for Package/Class/ Interfaces (borders).

This preference allows you to choose the color of borders inside your diagram.

Eclipse presentation.



Eclipse presentation if Use of colors for borders is selected in the menu.

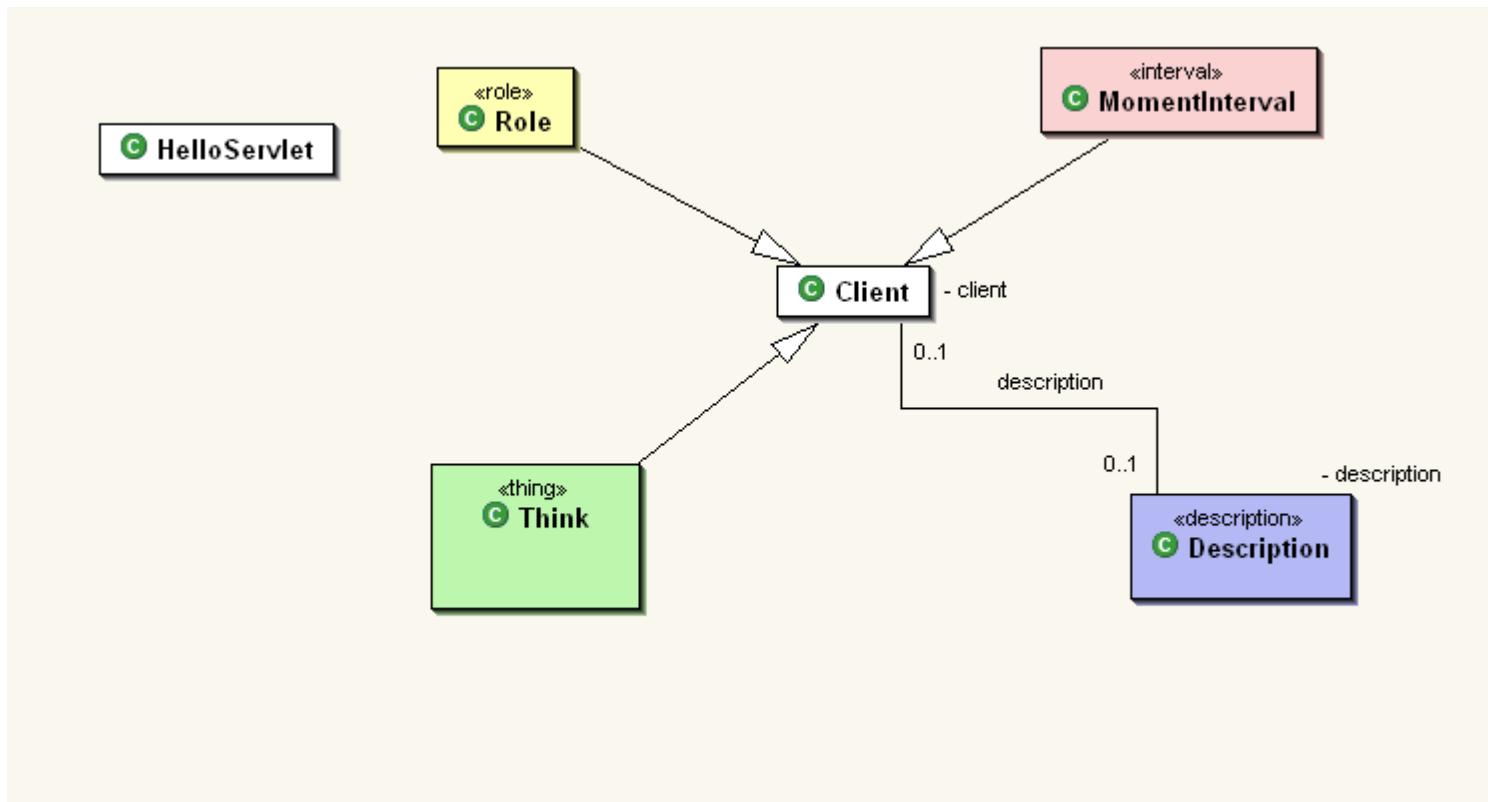


3. Use colors for Archetypes

This preference allows you to put colors depending on Archetypes in your Class Diagram Editor .

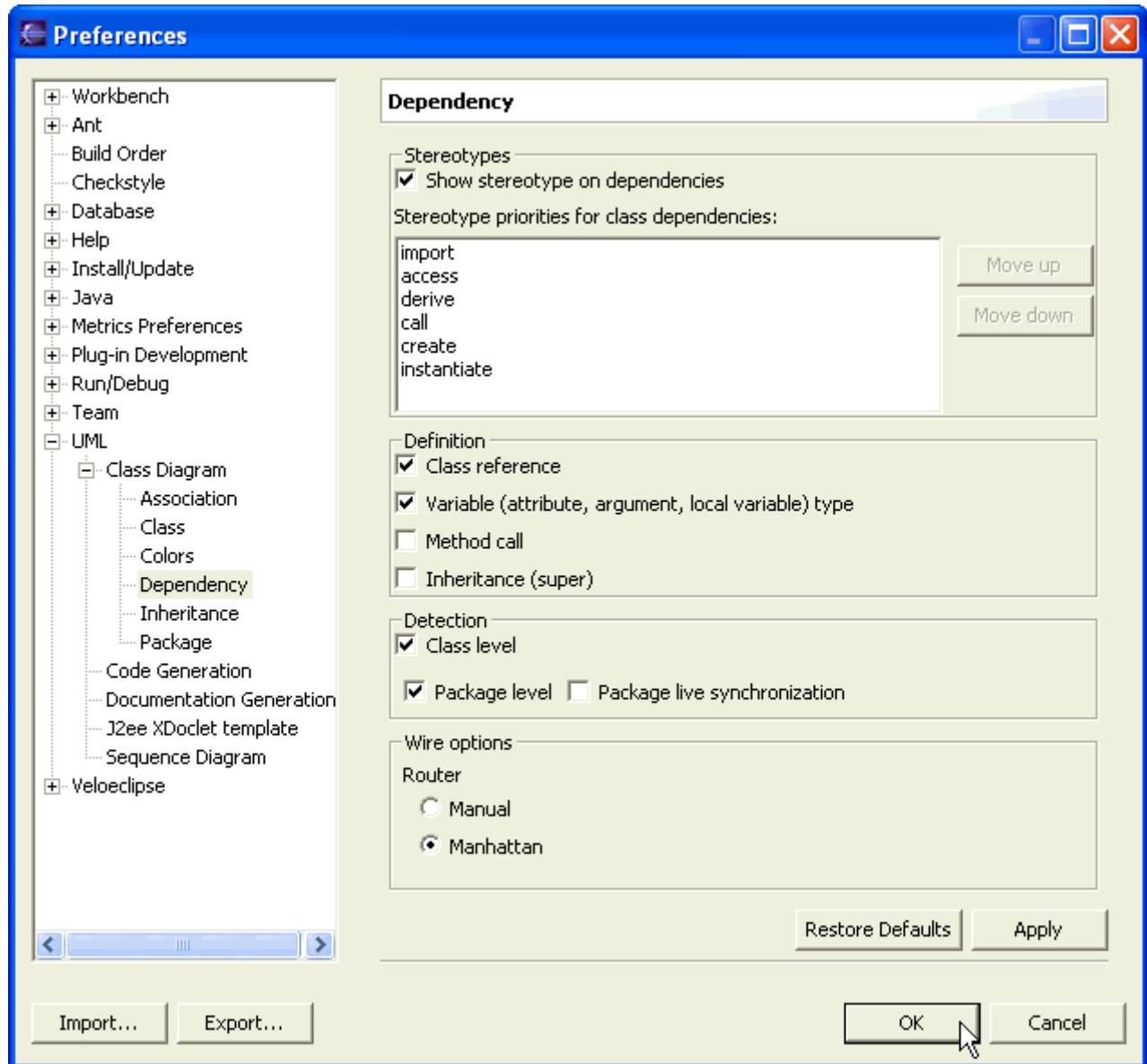
For possibilities are provided :

- Yellow: the value of the stereotype is *role*.
- Pink: the value of the stereotype is *moment or interval*.
- Green: the value of the stereotype is *thing*.
- Blue: the value of the stereotype is *description*.



Dependency

Dependencies can be customized from the preferences (UML > Class Diagram > Dependency).



Different options are available:

1. The Stereotype area:

Either **show or not show the stereotype** on dependencies

Select **stereotype priorities** for class dependencies

EclipseUML will just show one stereotype on a class dependency, even if a class dependency could have more than one stereotype. We will therefore have to select a priority order within the six kinds of stereotype. This priority order will decide which stereotype will be shown.

The six kinds of stereotype are (considering two elements A and B):

- **import:** B is imported by A
- **access:** A uses a B attribute
- **derive:** A derives from B
- **call:** A uses a B method
- **create:** A is a B factory
- **instantiate:** A instantiates B

2. Dependency definition area

This preference allows you to define the dependency relationships:

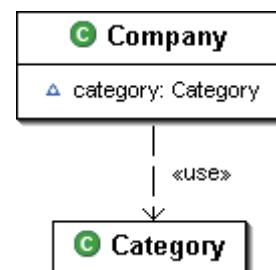
Options	Stereotypes	Description
Class reference	«access»	Access directly class attributes and call static methods.
	«instantiate»	Create an instance
Variable type	«access»	All variable types used in a class/interface such as attribute type, method argument type, method return type and local variable type.
Method call	«call»	Invoke an instance method
Inheritance (super)	«derive»	Specification or implementation of one type

This definition is used to detect the dependency relationships once a Java class is modified.

Class reference

Direct class attribute access

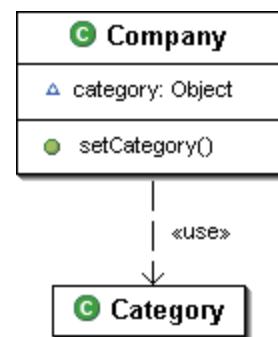
```
public class Category {
    static public Category software = new
    Category();
}
```



```
public class Company {
    Object category = Category.software;
}
```

Instance creation

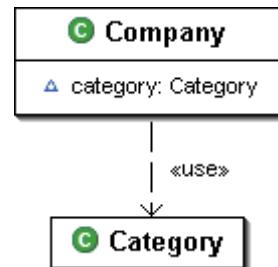
```
public class Category {  
}  
  
public class Company {  
    Object category = new Category();  
}
```



Variable type

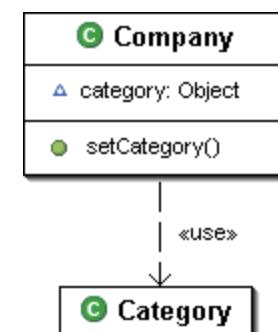
Attribute type

```
public class Category {  
}  
  
public class Company {  
    Category category;  
}
```



Method argument type

```
public class Category {  
}  
  
public class Company {  
    Object category;  
  
    public void setCategory(Category  
category)  
    {  
        this.category = category;  
    }
}
```

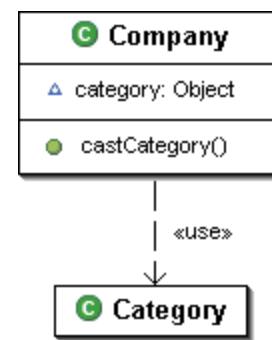


Method return type

```
public class Category {
}

public class Company {
    Object category;

    public Category castCategory()
    {
        return (Category) category;
    }
}
```



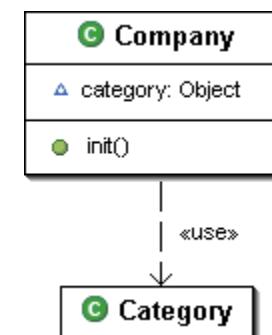
Local variable type

```
public class Category {
}

public class Company {
    Object category;

    public void init()
    {
        Category defaultCategory = null;

        this.category =
defaultCategory;    }
}
```



Method Call

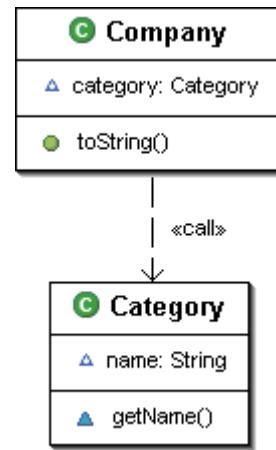
The dependency based on the method call is setup between the caller and the method declaration class.

```

public class Category {
    String name;

    String getName() {
        return name;
    }
}

```



```

public class Company {
    Category category;

    public String toString() {
        return category.getName();
    }
}

```

The following example doesn't fit this condition since the method `toString()` isn't defined the class `Category`.

```

public class Category {
}

```



```

public class Company {
    Category category;

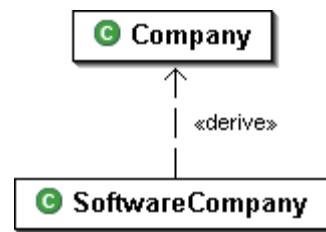
    public String toString() {
        return category.toString();
    }
}

```



Inheritance

```
public class Company {  
}  
  
public class SoftwareCompany extends  
Company {  
}
```



3. Dependency detection area

Two levels of dependency are supported: between classes and packages. The package dependency live synchronization needs extra CPU resources.

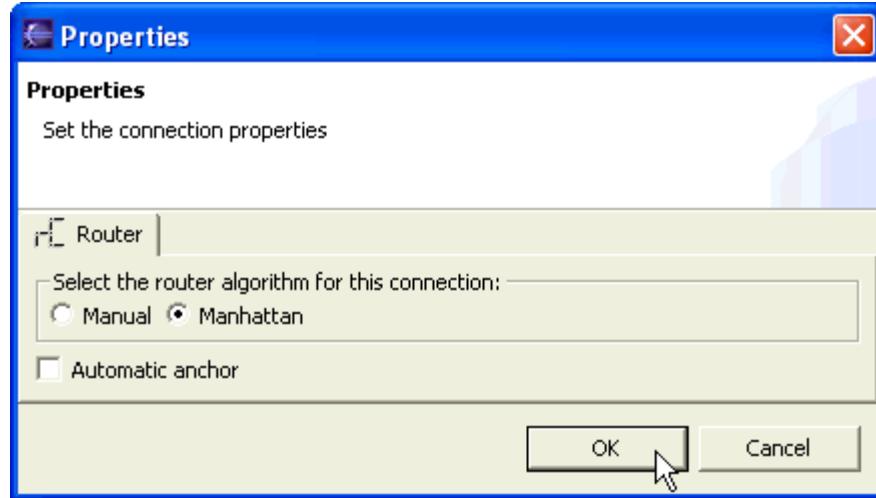
4. Wire options area

This preference allows you to choose the dependency default router and anchor.

- The router link could be manual or designed automatically using the Manhattan option

Inheritance

Associations can be customized from the preferences (UML > Class Diagram > Inheritance).



Wire options

This preference allows you to choose the inheritance default [router](#) and [anchor](#).

- The router link can be manual or designed automatically using the Manhattan option
- The anchor can be automatically fixed by selecting the, "Automatic anchor" checkbox

Package

The Omondo mechanism of Package preferences has two different levels: global and package. From the menu bar, Select **Window > Preferences > UML > Class Diagram > Package**

1. **The Eclipse preferences** concerning package are:
(first Level)

- In the **stereotype** area, you can decide to either show or not show the stereotype on packages
In the Package name format area, you can decide either to have:
 - a short name

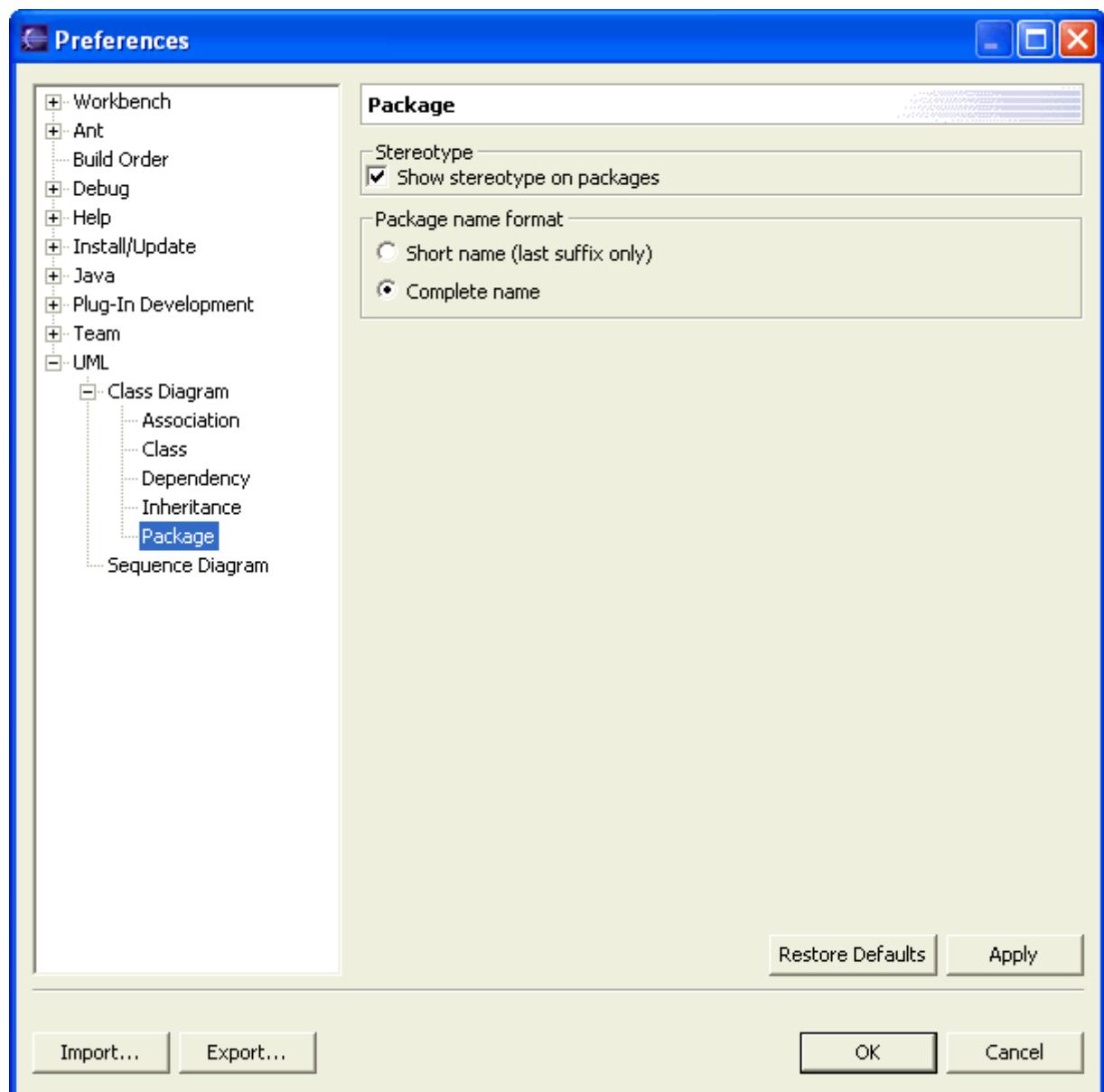


- o or a complete name



According to the general [EclipseUML preference mechanism](#), these preferences will not change the existing diagram. Existing diagrams can be changed using the diagram editor preferences. Using the Apply button allows you to change the existing package's preferences, while clicking on the OK button will just validate the preferences for new packages.

Here is what the Package preferences page looks like:



2. The class diagram preferences concerning package are:
(second Level)

- In the **stereotype** area, you can decide to either show or not show the stereotype on packages
- In the Package name format area, you can decide either to have a short or a long name.

You can work at class diagram level using the diagram editor's popup menu.

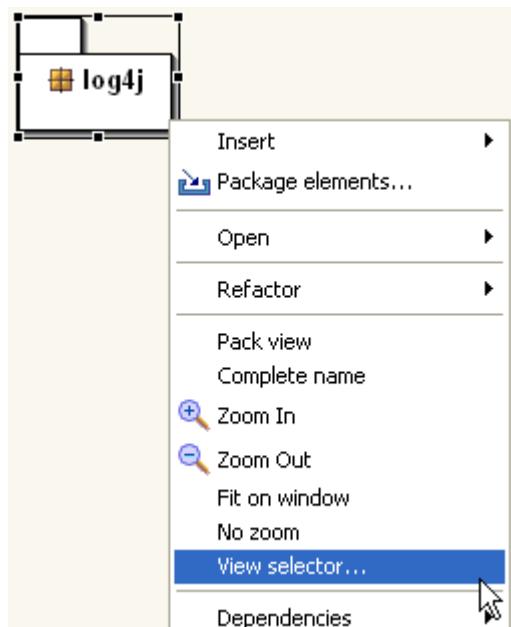
Open your class diagram editor and click inside the diagram. In order to work at global diagram level, be sure not to select any element.

3. The Package preferences concerning package are:
(third Level)

- In the **stereotype** area, you can decide to either show or not show the stereotype on packages
- In the Package name format area, you can decide either to have a short or a long name.

You can work at package level by selecting one package then right click to open the popup menu and finally choose View selector.

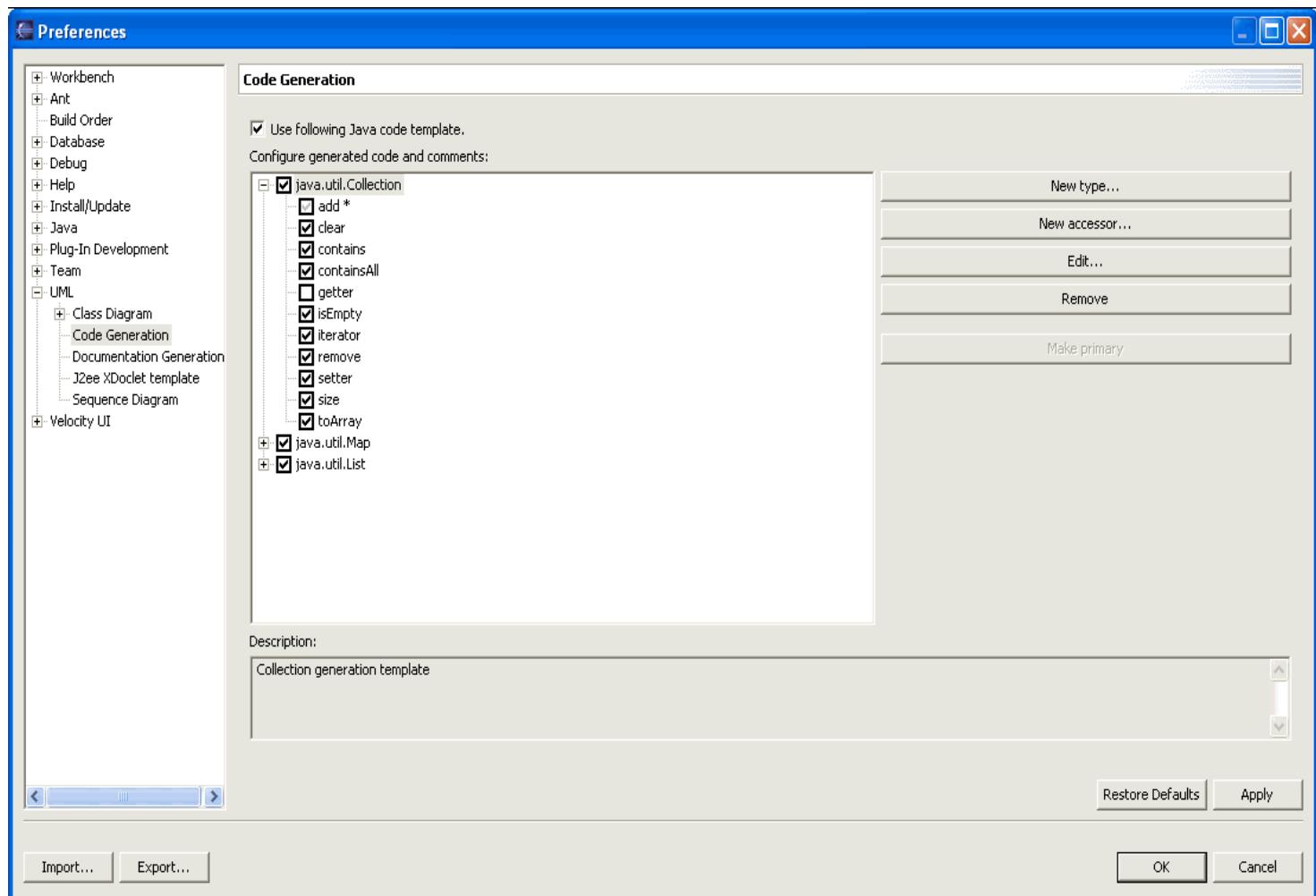
Here is the example:



Code Generation

The **code generation preferences** are used for customizing the EclipseUML code generated. The following options are available:

- Use Java code templates
- Configure generated code and comments
 - Add New type
 - Add New accessor
 - Edit
 - Remove
- Check the generation description

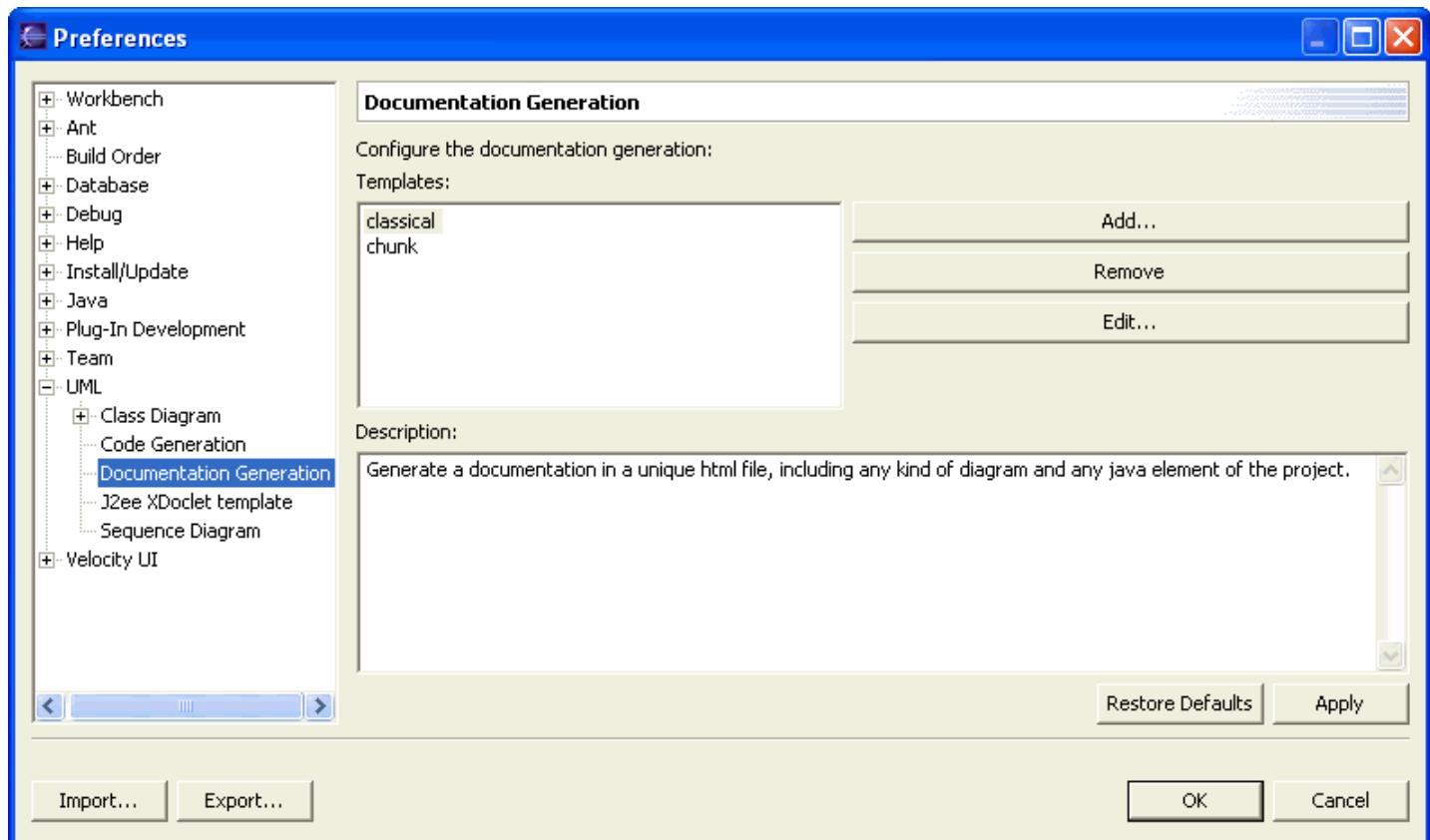


Documentation Generation

The documentation generation preferences are used for customizing the EclipseUML project documentation generated.

The following options are available:

- Configure the documentation generation using templates
 - Add
 - Remove
 - Edit
- Check the generation description



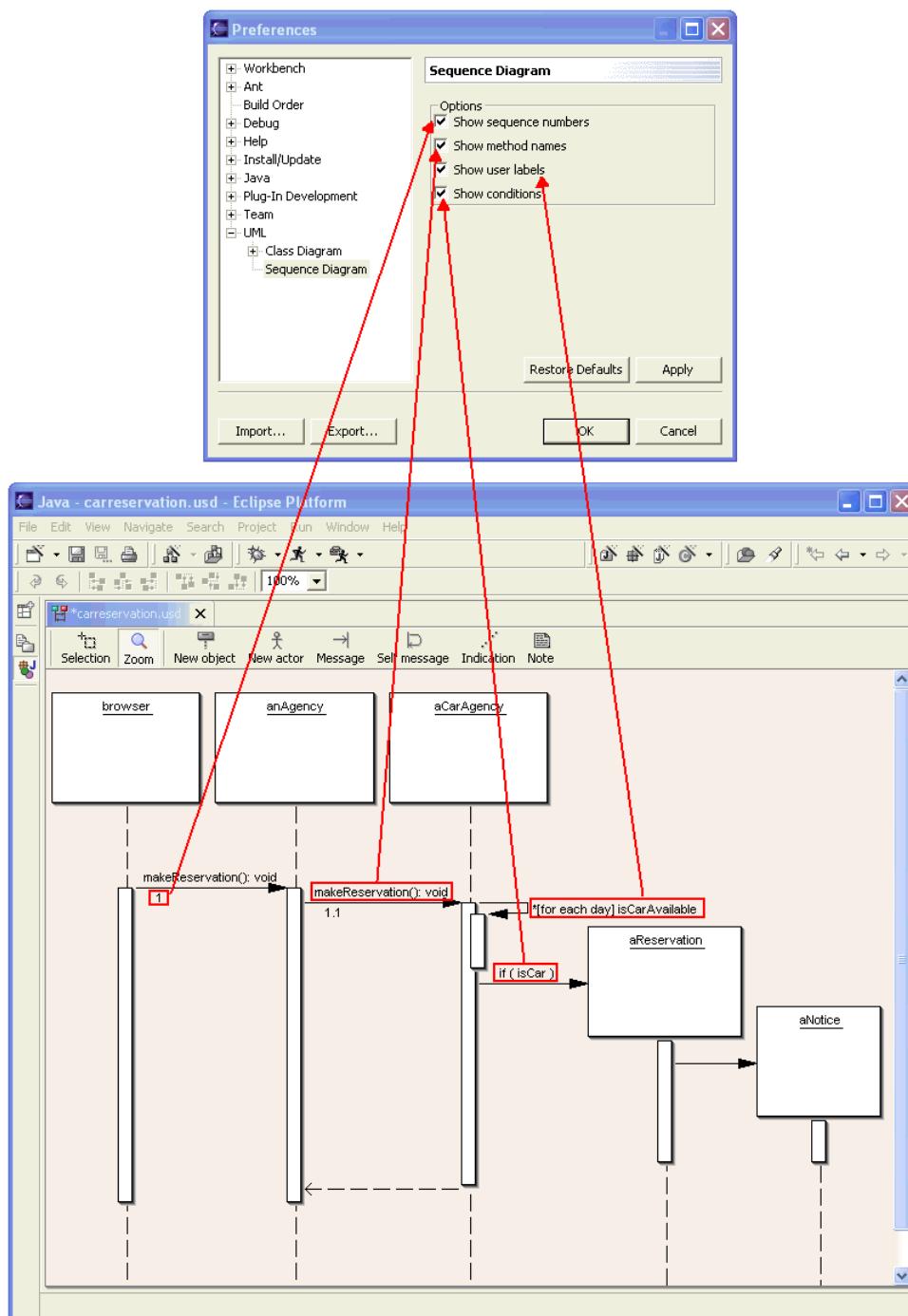
Sequence Diagram

The Sequence Diagram Preferences allow you to choose sequence diagram options. Four options are available and will help the modeler to show what is really important inside their diagram.

1. Show sequence numbers
2. Show method name
3. Show user labels
4. Show conditions

We can select each option in the Sequence Diagram window and click on Apply button in order to see the live result in the diagram editor.

The following pictures show the sequence diagram preferences options and their relation to the sequence diagram editor.



J2ee XDoclet templates

This Preferences window allows you to customize the XDoclet tag templates that will be used in J2ee wizards.

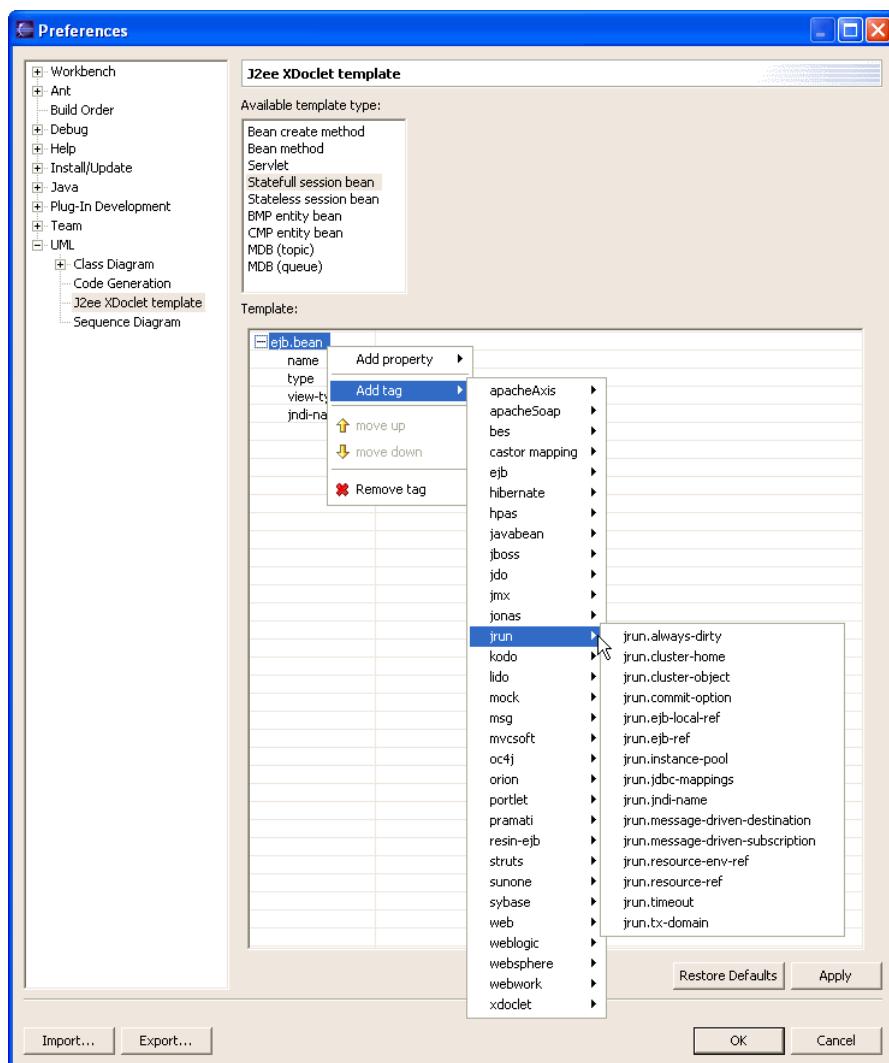
- 9 templates are available:

```

Bean create method
Bean method
Servlet
Statefull session bean
Stateless session bean
BMP entity bean
CMP entity bean
MDB (topic)
MDB (queue)

```

We can select each template in the right pane inside the Available template type list. Clicking on a specific template will load the template content into open the template editor. Inside the template editor use the popup menu to add/remove tags and tag properties. The following window shows how to add a tag:



J2ee Profiles

Concept

A UML profile expresses the semantics of your application using a set of predefined extensions to UML. UML profiles allow developers to share a common graphical notation and vocabulary, and permit more precise specifications and better documentation on how to use and customize systems. In this J2ee profile we will present the Class and Deployment Diagram integration using specific stereotypes. We will also use this profile to extend your application from modeling to real Web modules deployment using JBoss, Tomcat or any J2ee application server.

The J2ee Profile will be natively integrated inside EclipseUML Class diagram using the following stereotypes:

1. Class stereotypes

- servlet
- bean

2. Method stereotypes

- create
- business

The J2ee Profile will be natively integrated inside EclipseUML Deployment diagram using the following stereotypes on components:

- ear
- war
- jar
- sevlet
- bean

EclipseUML UML profiles work at project level.

Getting Started

In this section you will learn how to get started using J2ee EclipseUML profile.

You first need to create a Java project. For the purpose of our documentation, we will create a HelloWorld project.

Be sure that you have downloaded all the libraries necessary to develop Servlets and EJB. These libraries can usually be found in the application server lib directory or downloaded from the Sun website.

The Getting started section covers the following elements:

1. [Add required Jars](#)
2. [Set up UML profile properties](#)

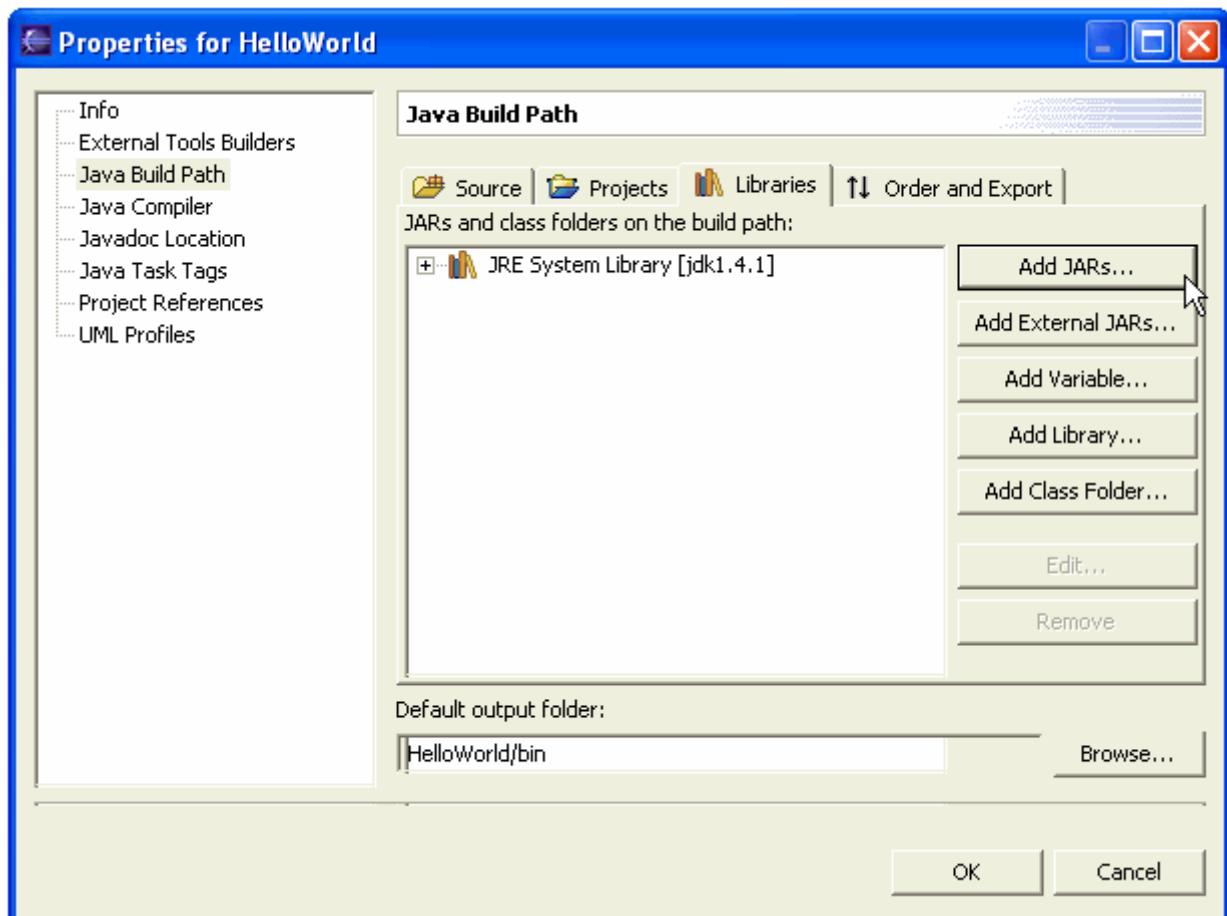
1. Add required Jars

Create and Select a project in the Package Explorer.

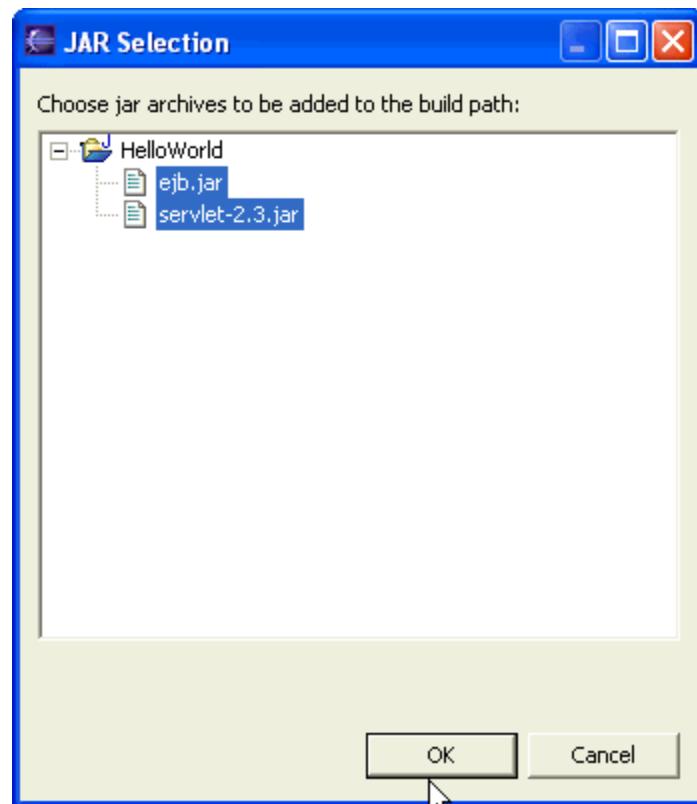
By right-clicking open the popup menu > **Properties** > **Java Build Path** and select **Libraries** in the upper right pane.

We need to add the servlets and EJB interfaces Jar files to the project classpath :

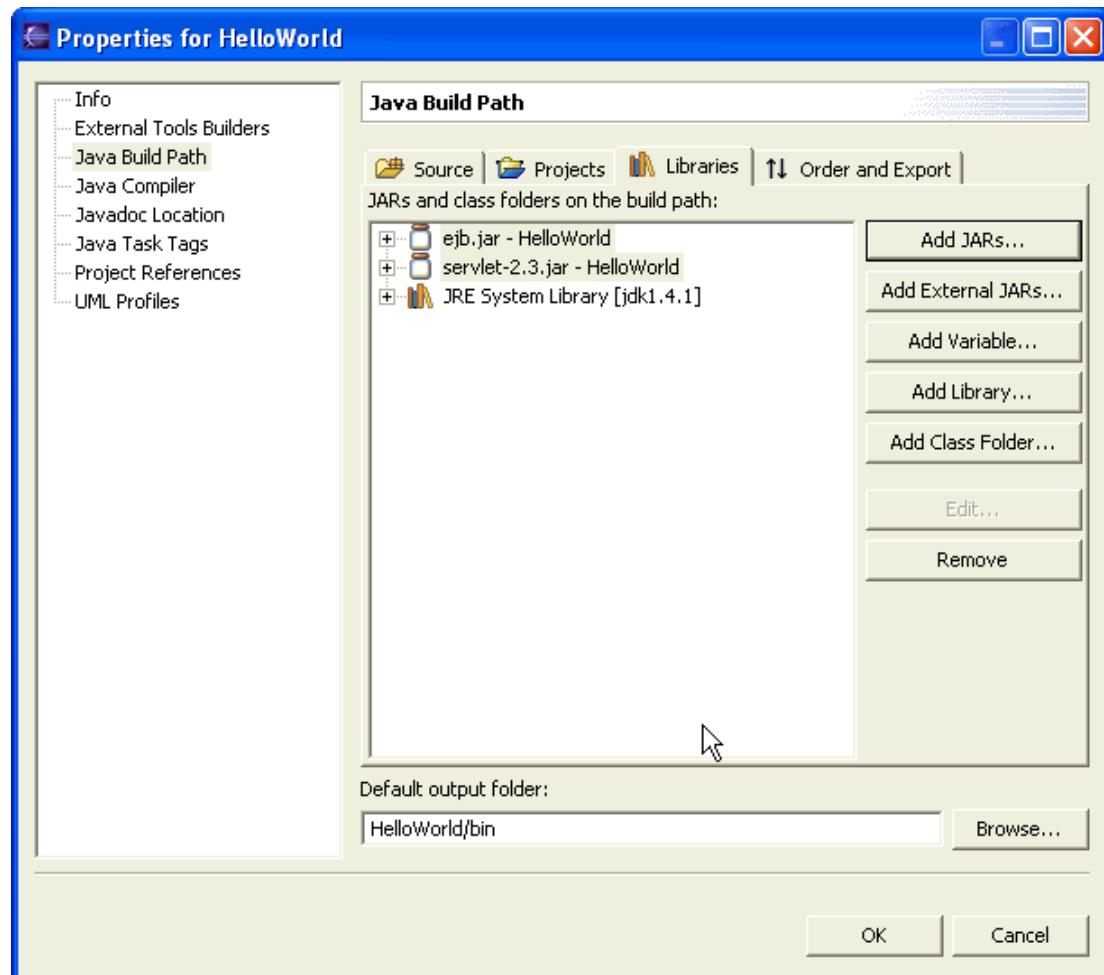
- Click on the *Add JARs...* button if the jar files are available in your project.
- Click on the *Add External JARs...* button if the jar files are not available in your project. These libraries can be found in the lib directory of the com.omondo.uml.std plugin.



Select Servlets and EJB home interfaces Jar files, and click on the OK button.



The selected Servlets and EJB interfaces Jar files have been added to your project.



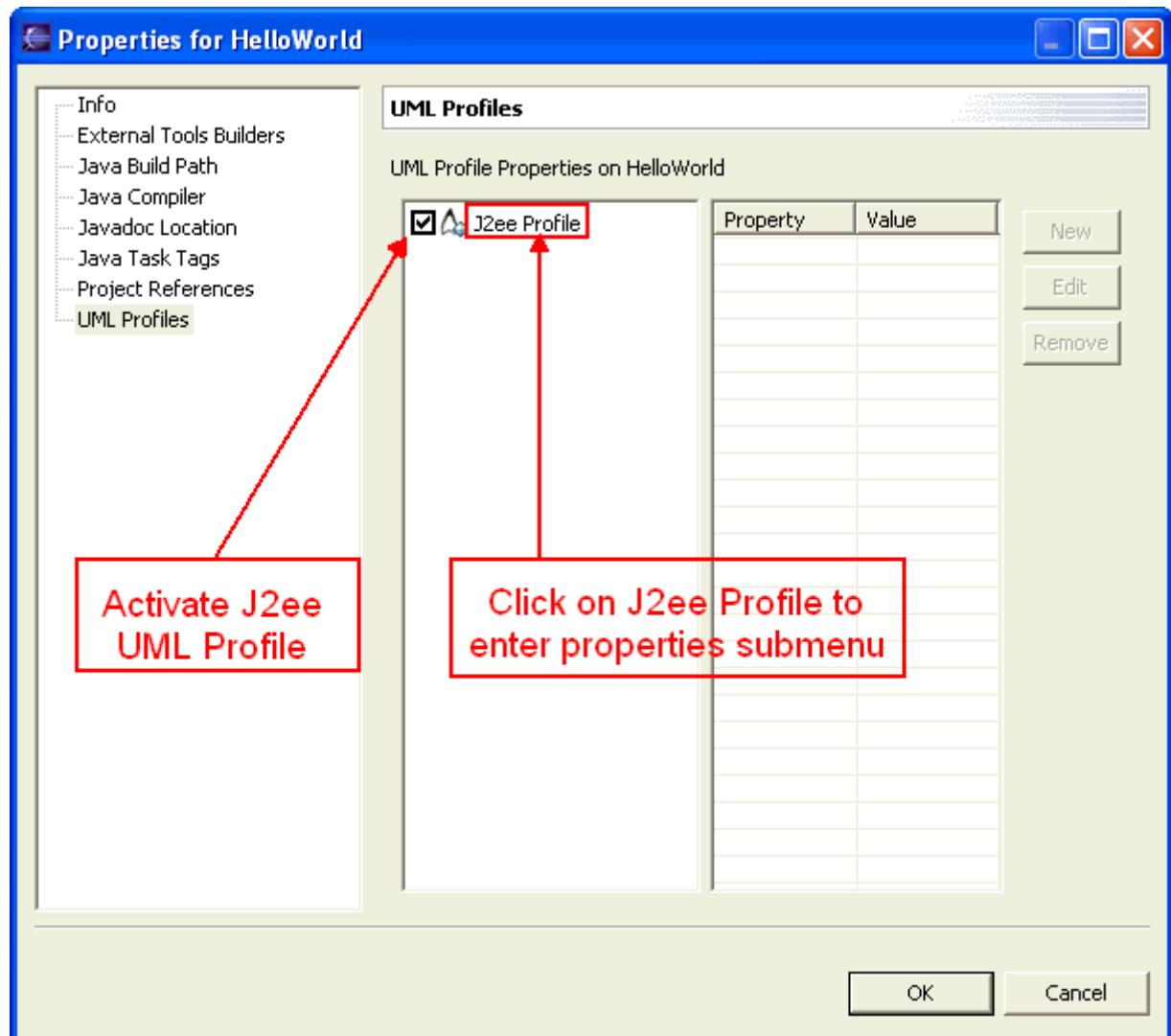
2. Set up J2ee profile properties

J2ee profiles will only be activated in your diagrams if you select them in the Project properties window.

We currently have one UML Profile, but other profiles will be added very soon.

Properties are specific to one project. This allows you to use different profiles inside EclipseUML depending on your modeling needs.

By right-clicking on a project in the Package Explorer window, open the popup menu > **Properties** > **UML Profiles** and select **J2EE Profile** in the right pane.

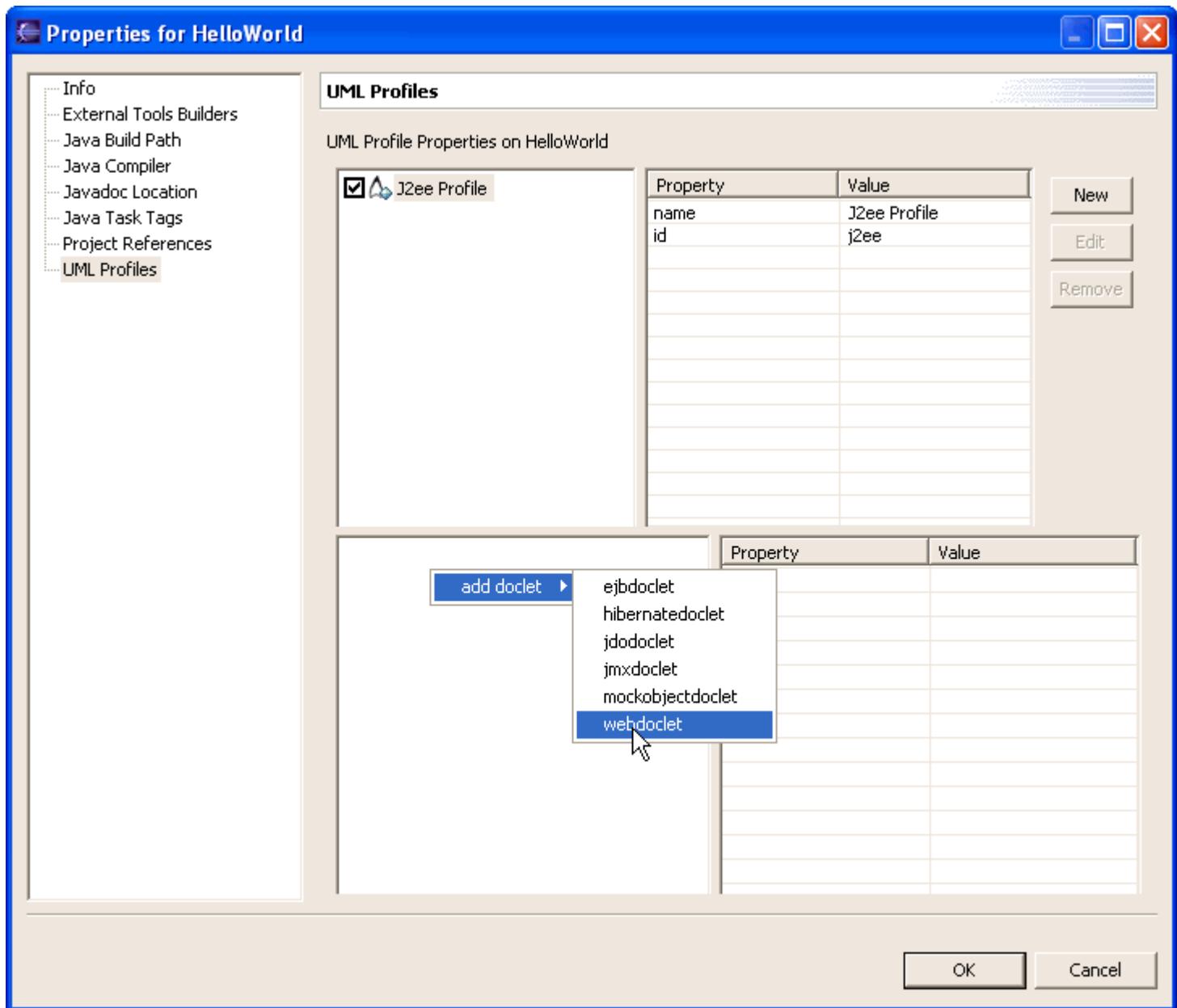


Open the J2ee Profile submenu and add the Doclets to use.

It is important to understand that you can add six different kinds of Doclet (ejb, hibernate, jdo, jmx, mocko and web). Each doclet has its own sub task and each sub task has its own properties, which can be customized.

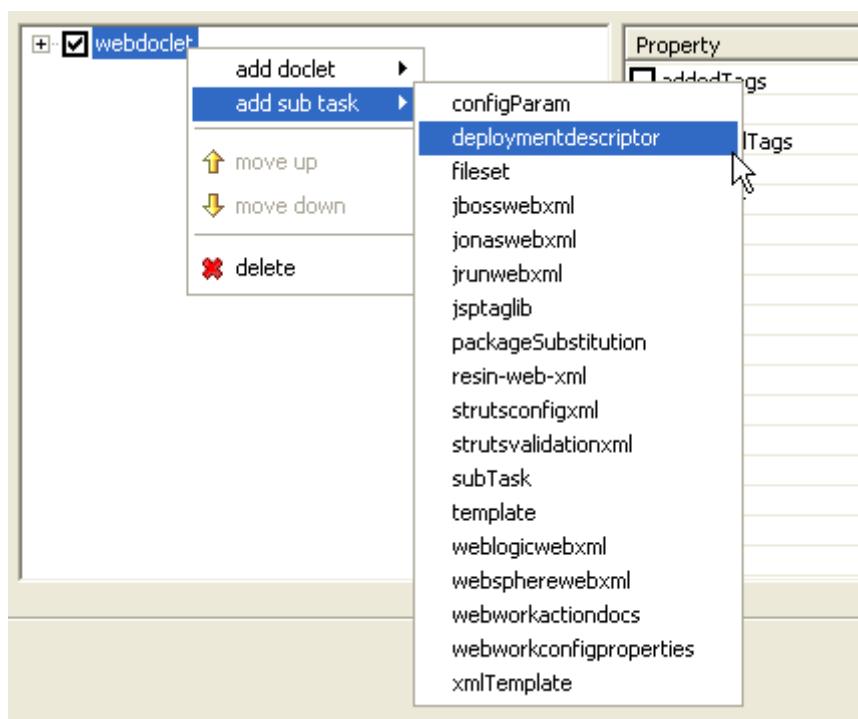
We need to select what kind of Doclet we are going to use.

For example, we can choose webdoclet. For more information, have a look at <http://xdoclet.sourceforge.net>

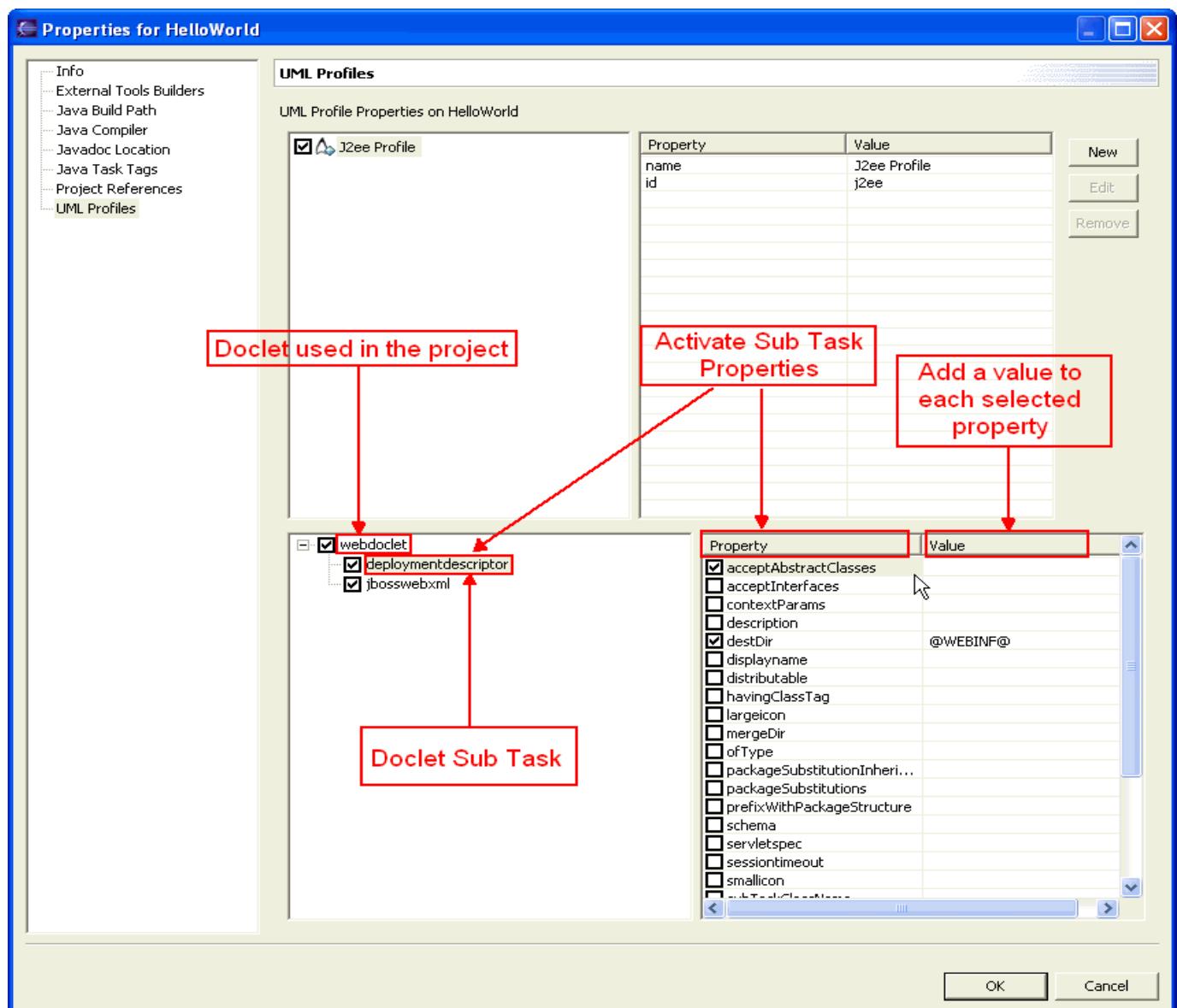


Once you select a Doclet, you can add new sub tasks to this Doclet. These sub tasks are specific to each Doclet. Select the Doclet in the menu and right click to open the **popup menu > add sub task > select a task**

For example, we can choose webdoclet and add a sub task named deploymentdescriptor.



When you have selected a specific sub task, you can then set its properties in the table on the right.



Creating Servlets and Ejb

In this section you will learn how to create Servlet and Ejb classes inside your Class Diagram. Do not forget that you will not be able to create Servlets and Ejb if you do not select J2ee profile in the project properties.

If you want to use the J2ee profile on existing Servlet or Ejb inside your Class Diagram, you must ensure that servlet classes have the 'servlet' stereotype and Ejb classes have the 'bean' stereotype. No other names are currently allowed if you wish to activate Servlet and Ejb XDoclet tab in the class properties dialog.

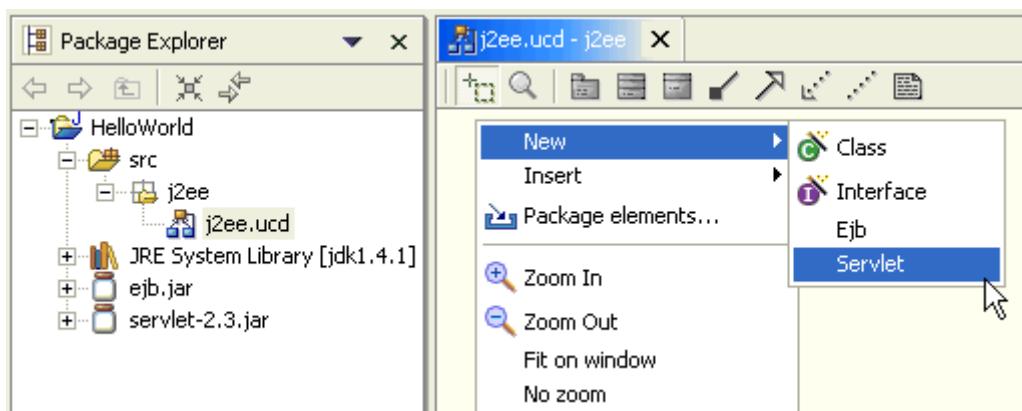
This section covers the following elements:

1. [Create a Servlet](#)
2. [Create an Ejb](#)
3. [Create an Ejb method](#)

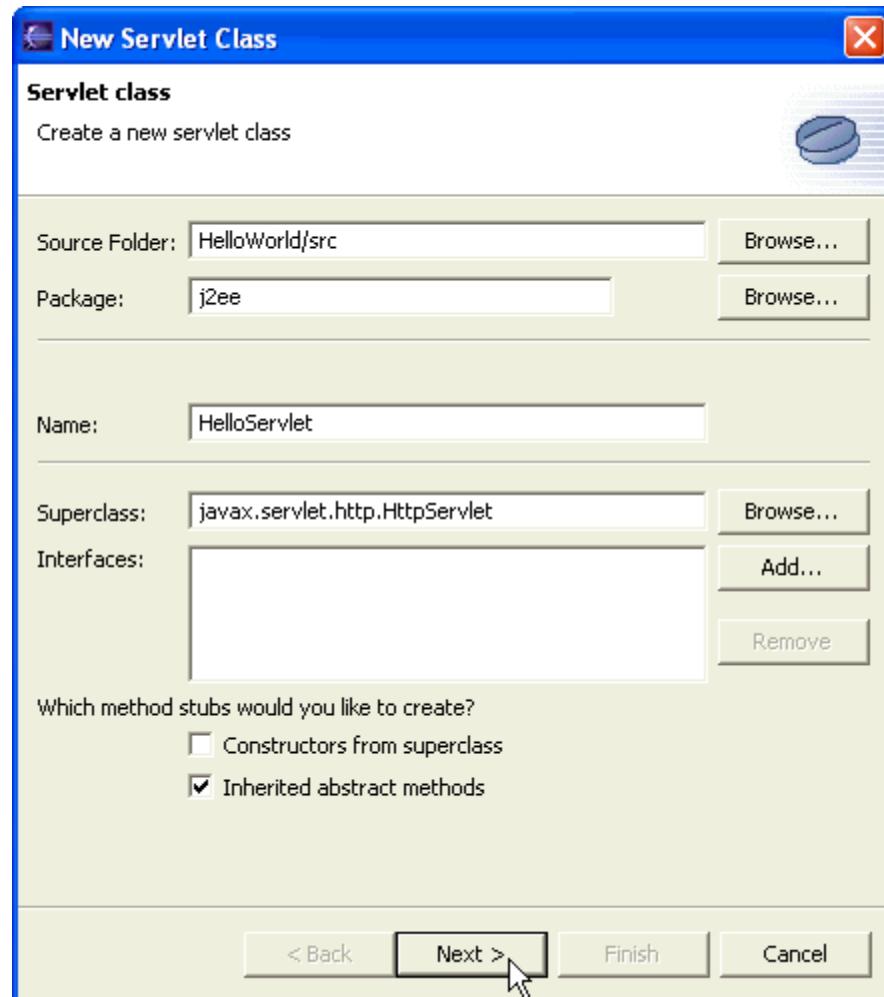
1. Create a Servlet

Create a new Servlet.

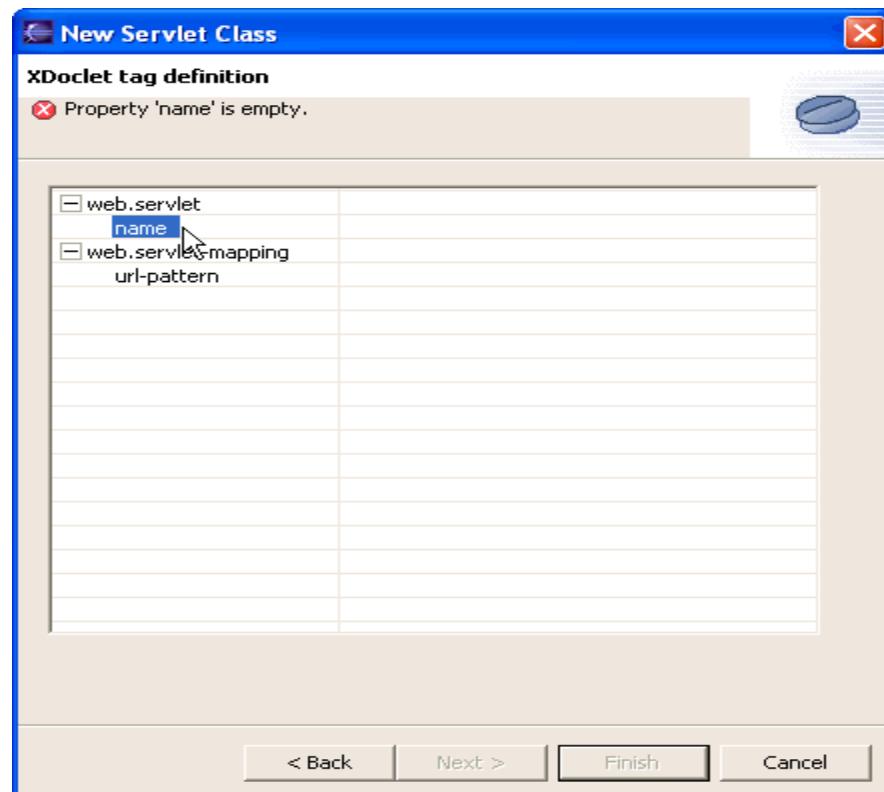
Use the **Class Diagram** popup menu > New > Servlet



Open the Servlet Creation wizard and set up Class properties.
Enter the Class name "HelloServlet" and click on the Next button.

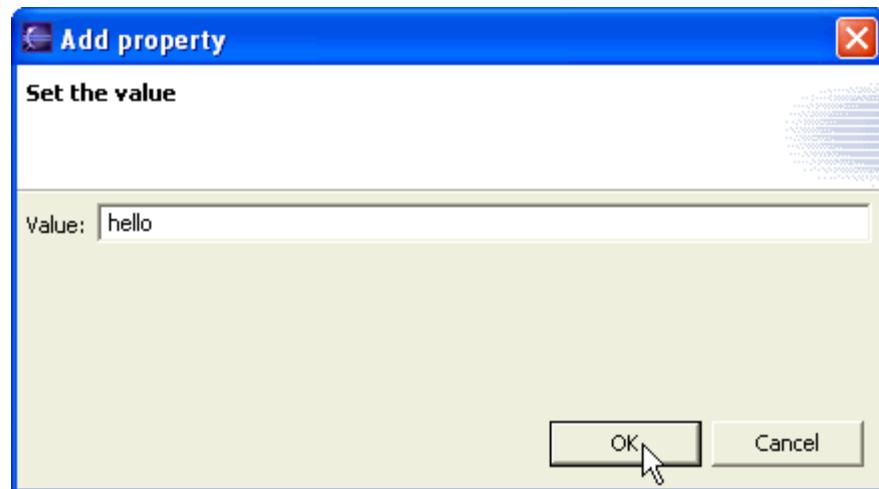


We can customize the default tags. These default tags come from the [J2ee XDoclet template preferences](#). Double clicking on empty tag properties will allow you to give them a value.



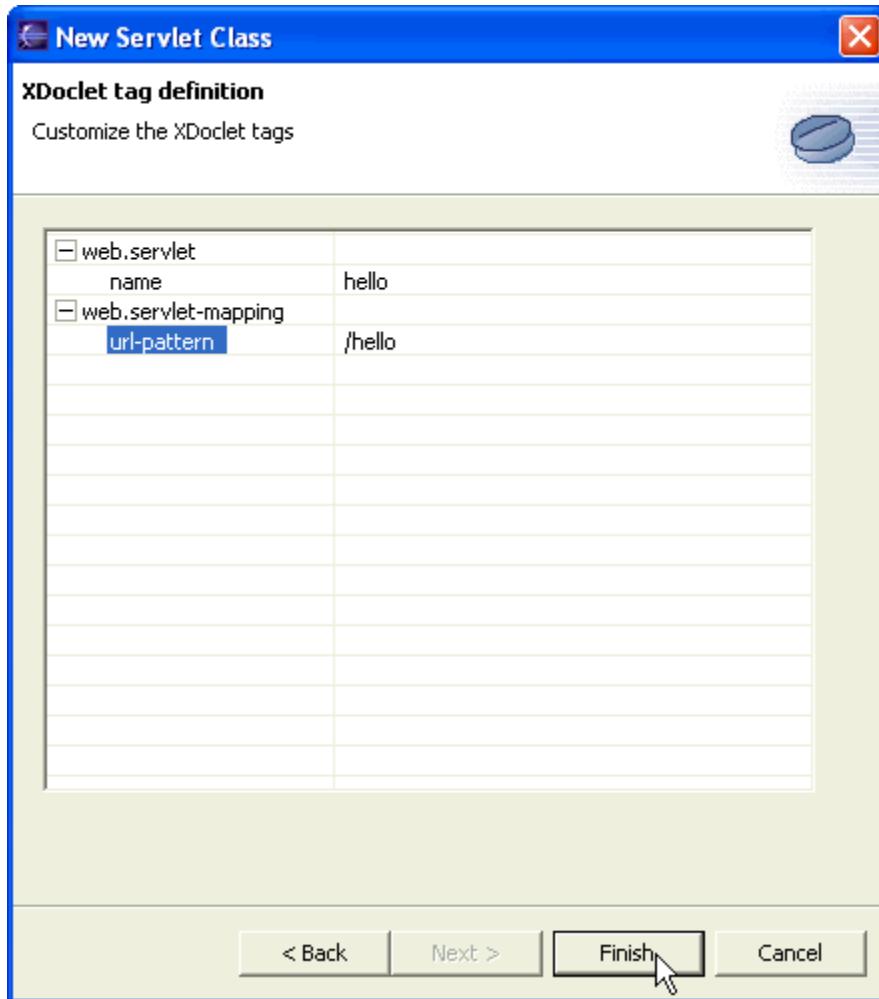
Enter the name of the tag.

Enter "hello" and click on the OK button.

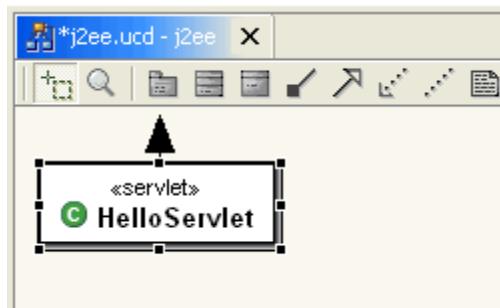


Enter the url-pattern name.

Enter "/hello" and click on the Finish button.

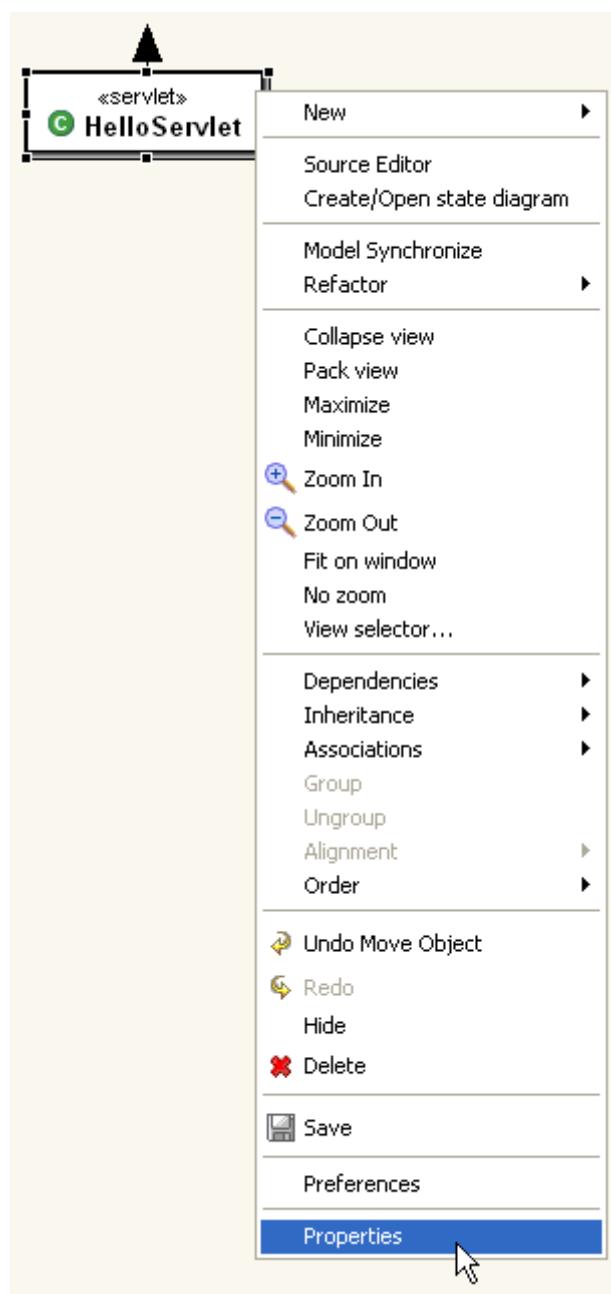


Our HelloServlet Class has been created and its stereotype is servlet.



We want to have a look at HelloServlet Class properties.

Select the HelloServlet element and right click, **open popup menu > properties**

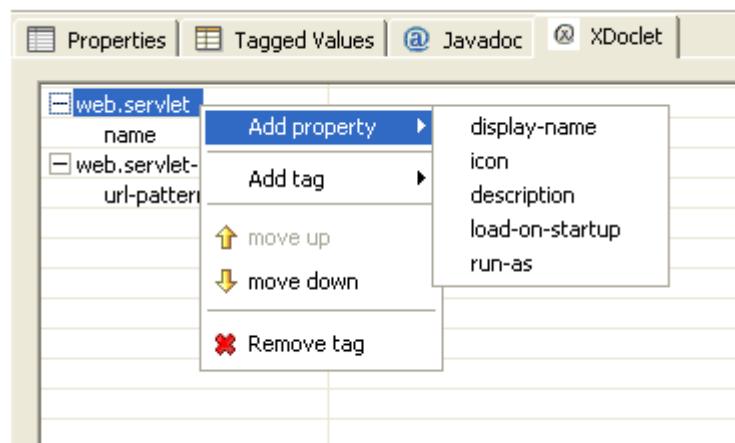


We want to see Xdoclet properties.

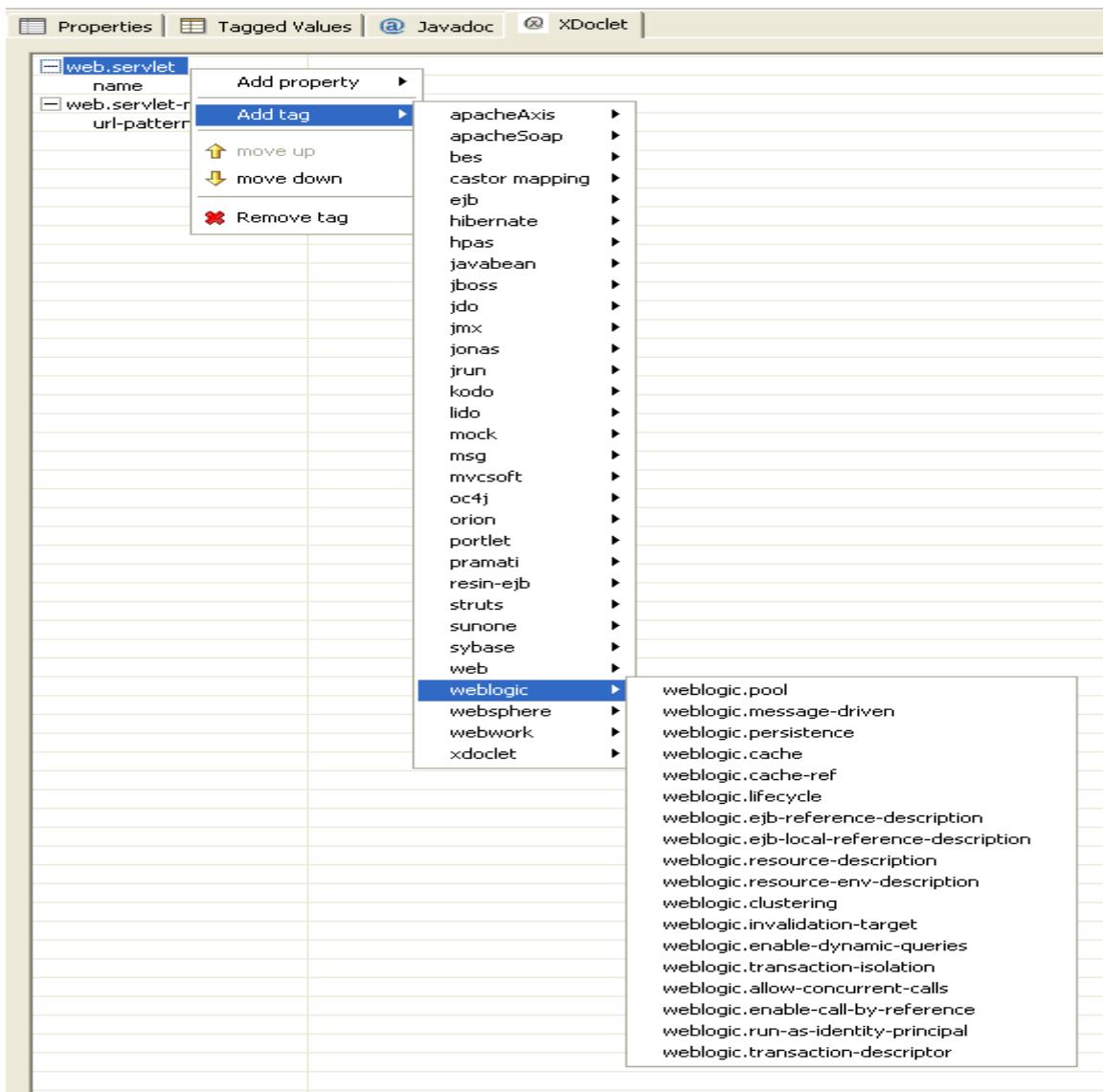
Select XDoclet tab in the Class properties window.

We can add a property and tag to each element.

The following window shows how to add a property to a XDoclet tag.:.



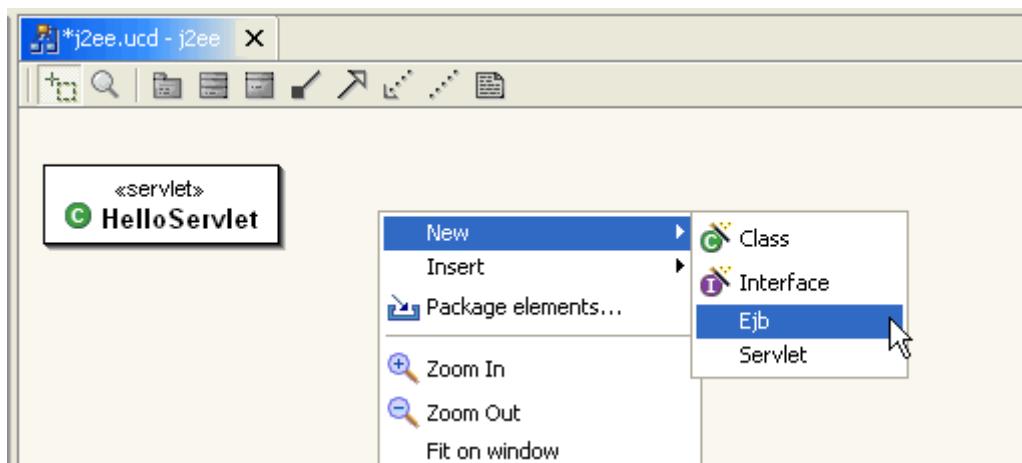
The following window shows how to add specific tags:



2. Create an Ejb

Create a new Ejb.

Use the Class Diagram popup menu > New > Ejb

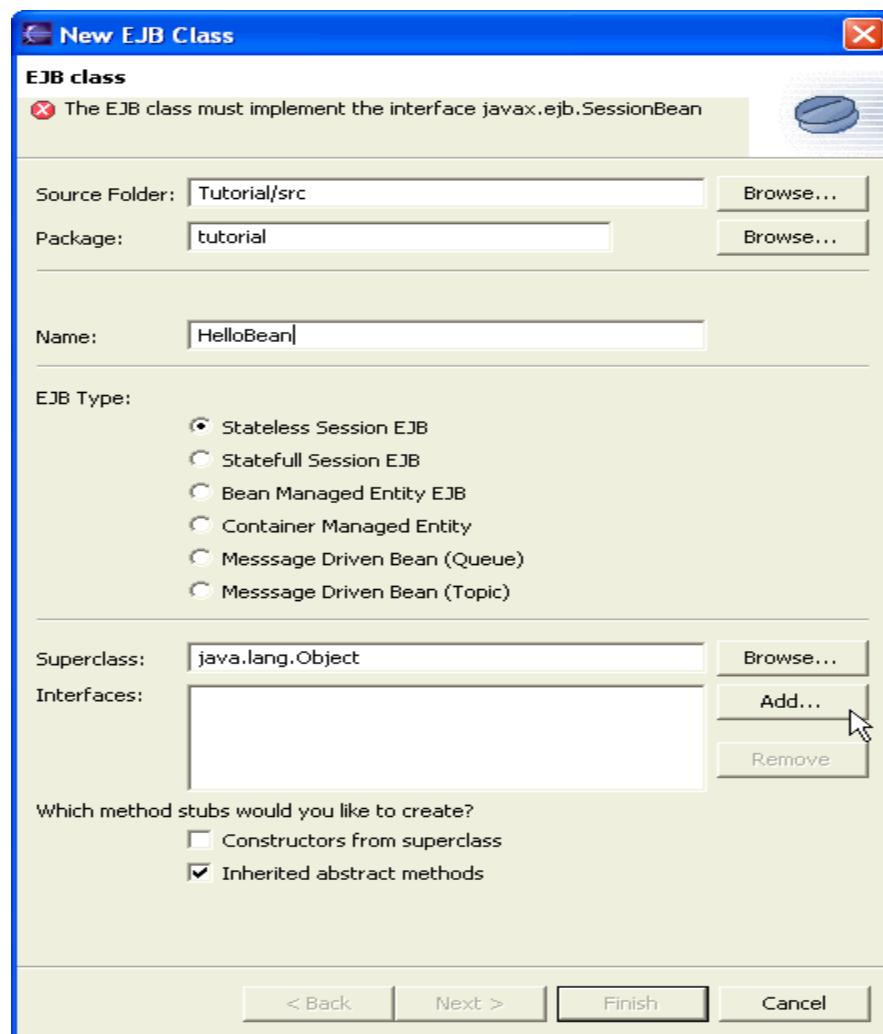


Open the Ejb Creation wizard and set up Class properties.

Enter the Class name "HelloBean" and add interfaces.

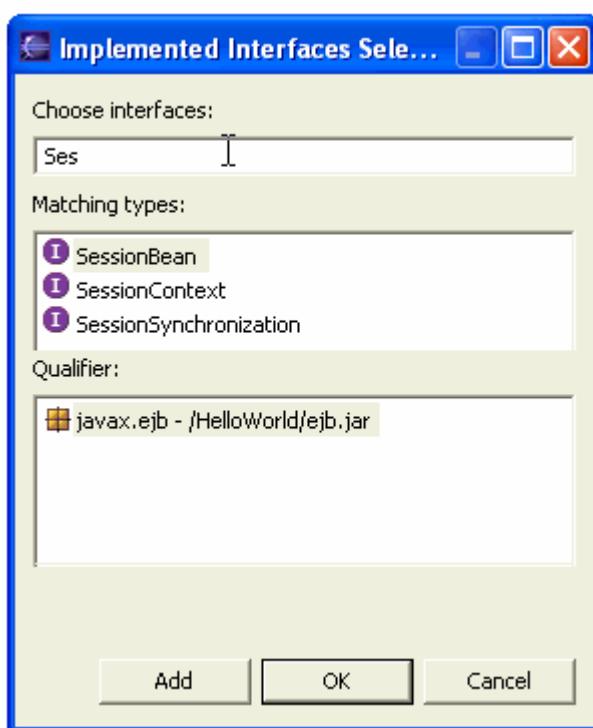
You will not be able to click on the next button till the correct interface has been chosen.

Click on the add button.



Select all required interfaces, by typing the first letter of the name in the Choose interface field and click on the Ok button.

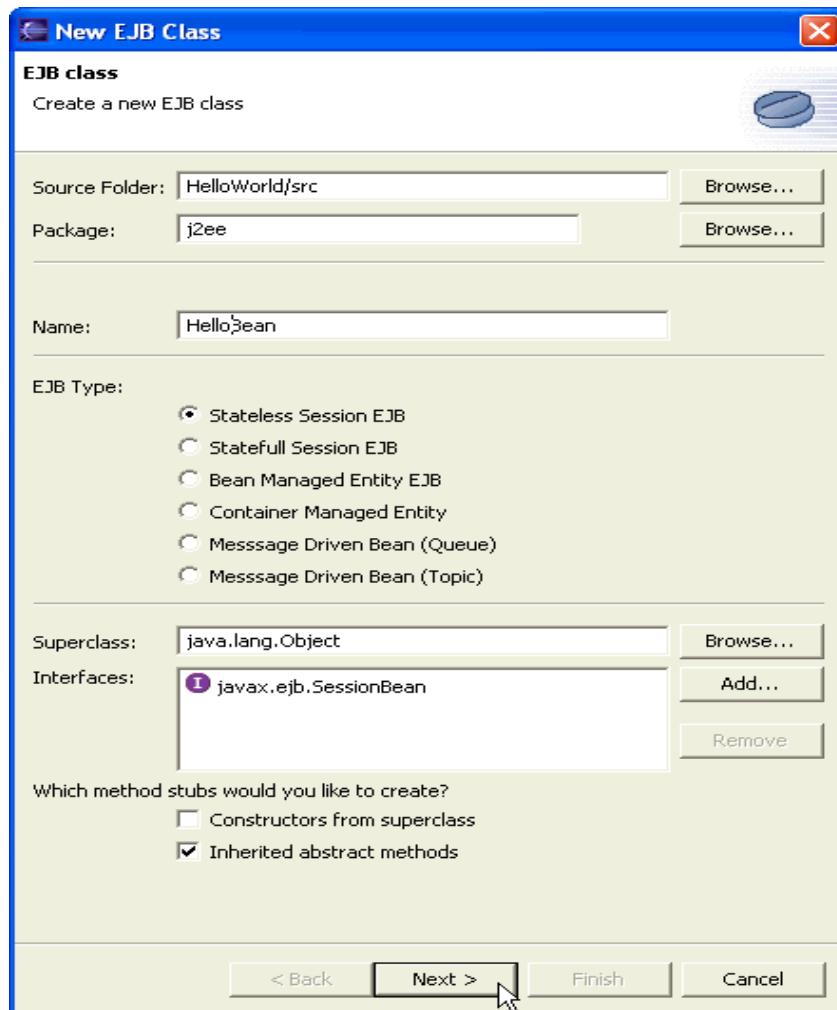
We selected "SessionBean".



The new Ejb Class is almost created, but we are still not ready to click on the Finish button.

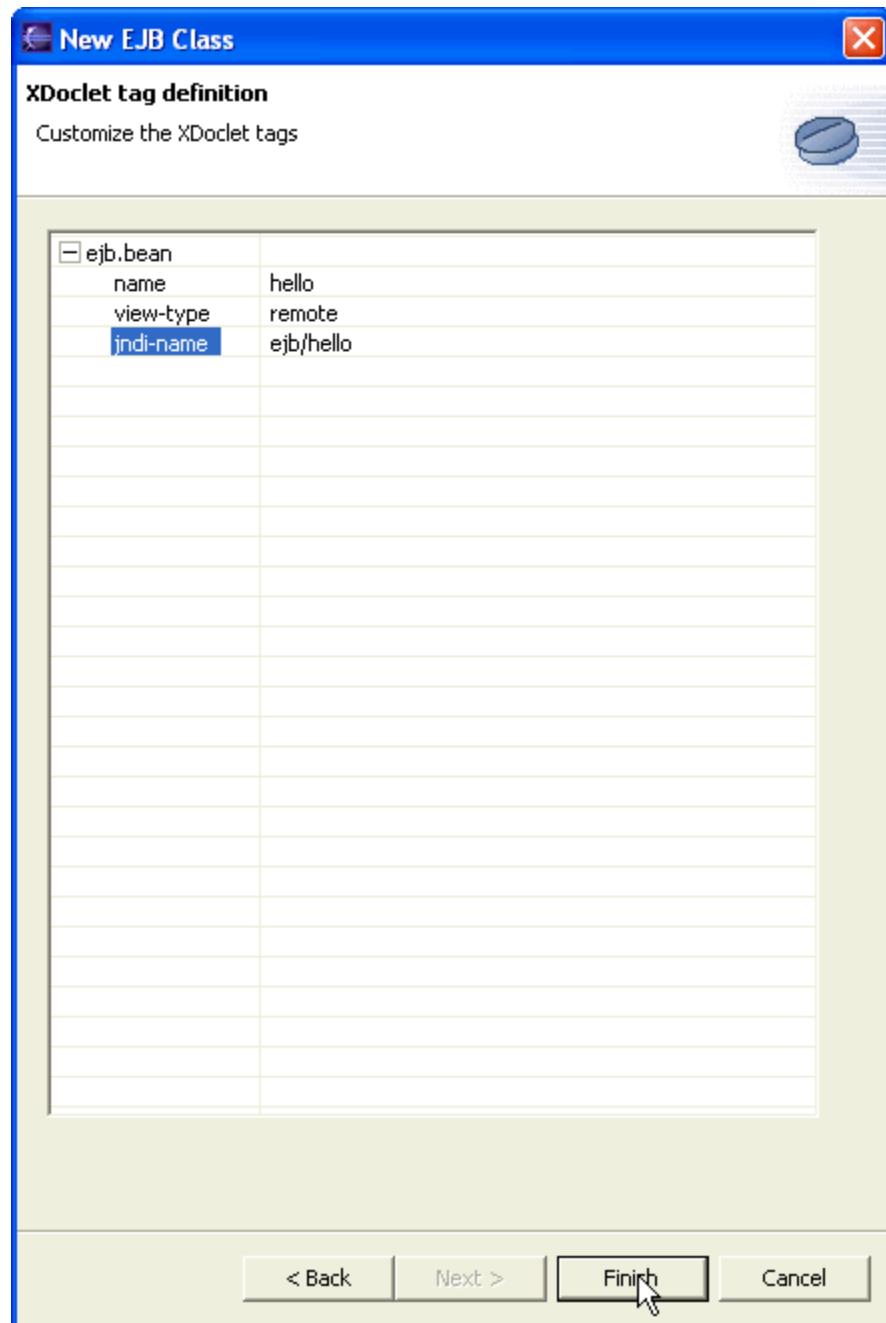
We decide that HelloBean will be a Stateless Session.

We click on the Next button.



We can customize the default tags. These default tags come from the [J2ee XDoclet template preferences](#).

Double clicking on empty tag properties will allow you to give them a value.



We want to have a look at EJB Class properties.

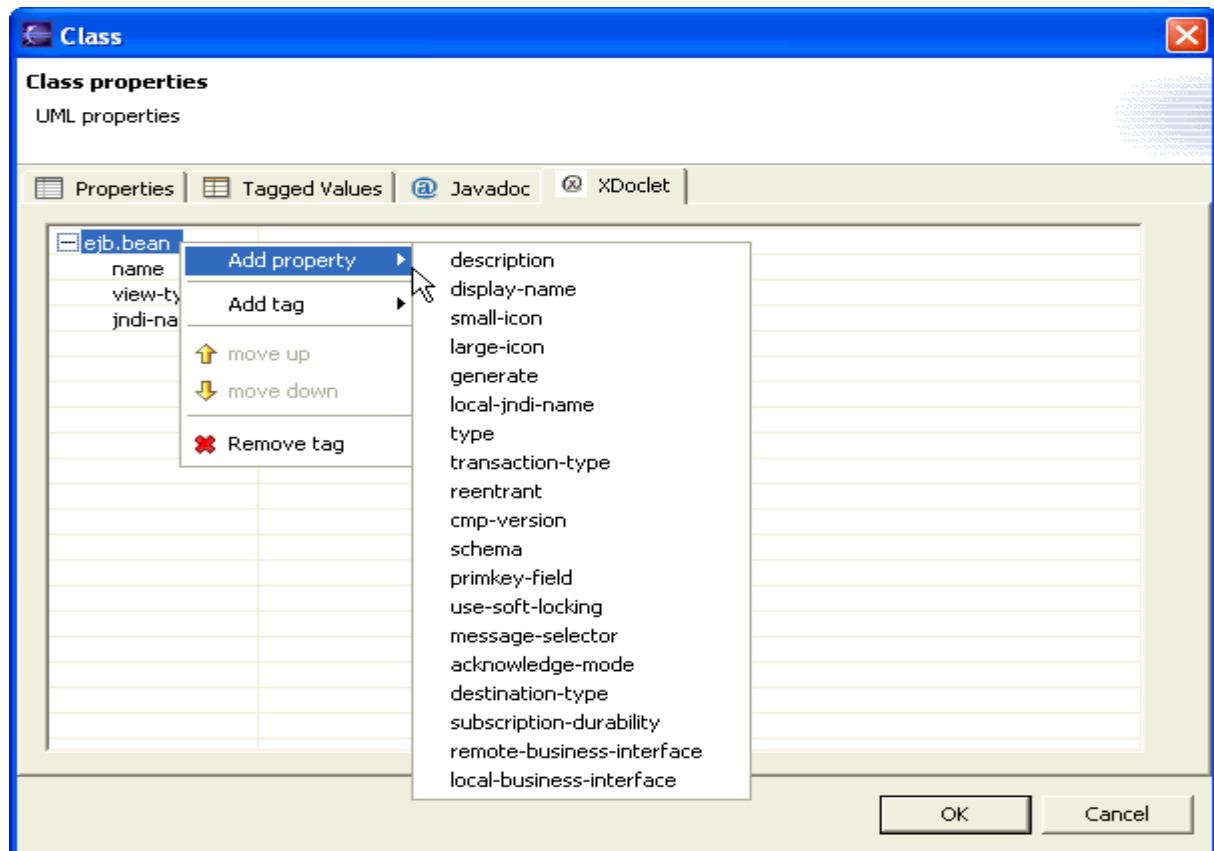
Select the EJB Class element and right click, **open popup menu > properties**.

We want to see Xdoclet properties.

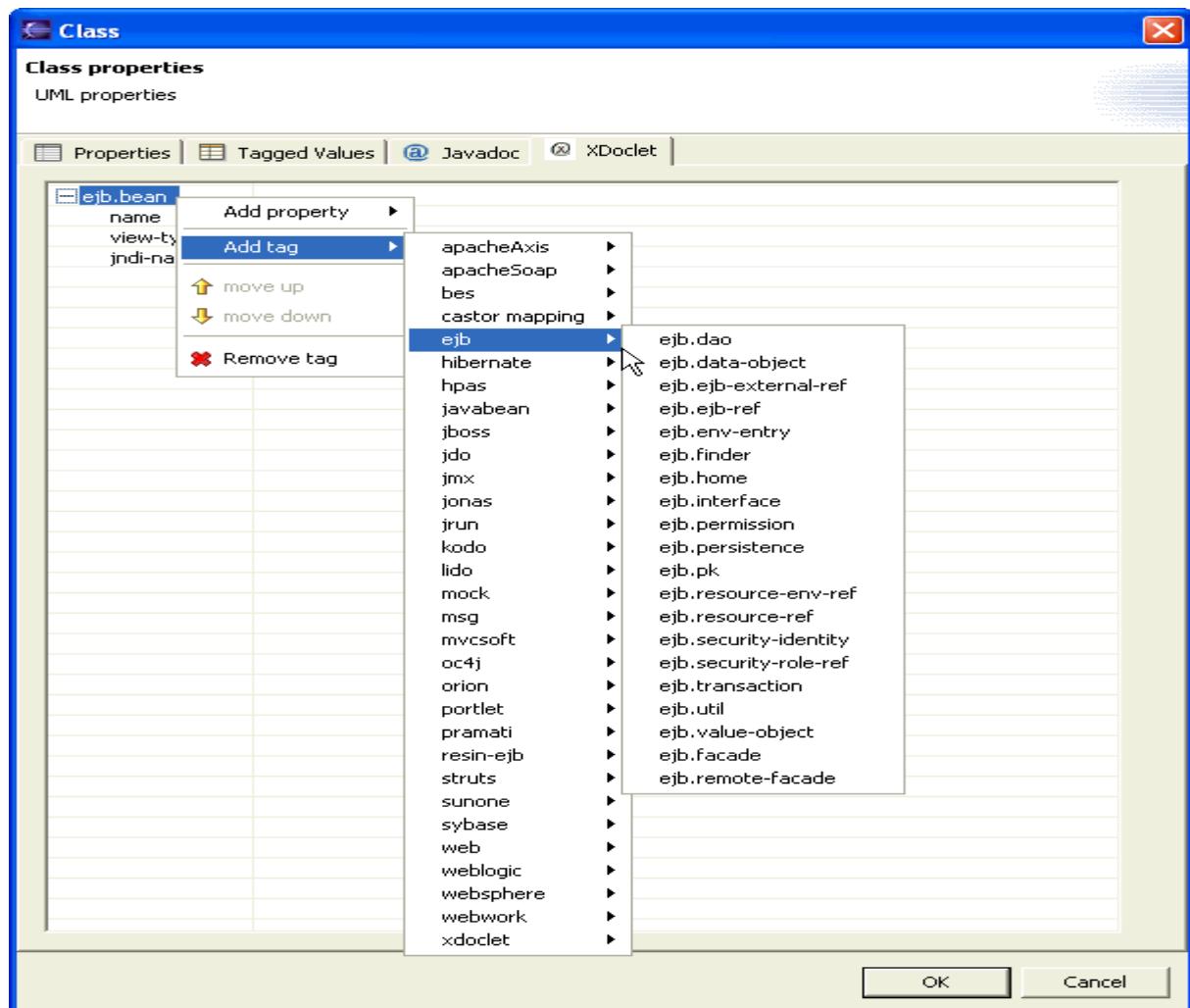
Select XDoclet menu in the Class properties window and click.

We can add a property and tag to each element.

The following window shows how to add properties:

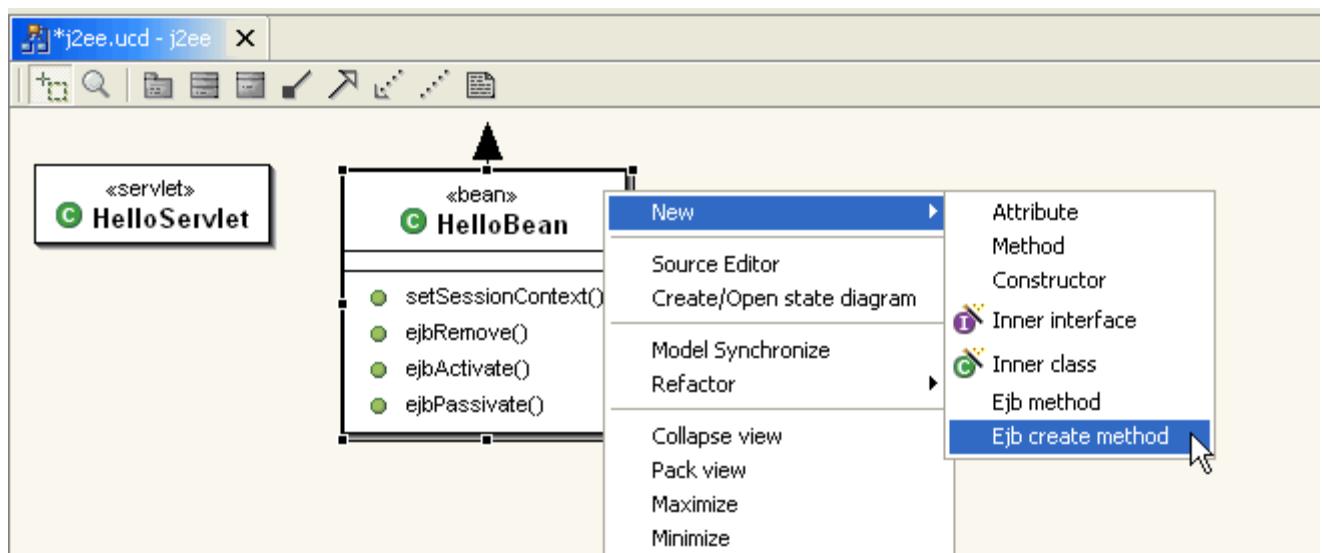


The following window shows how to add specific tags:

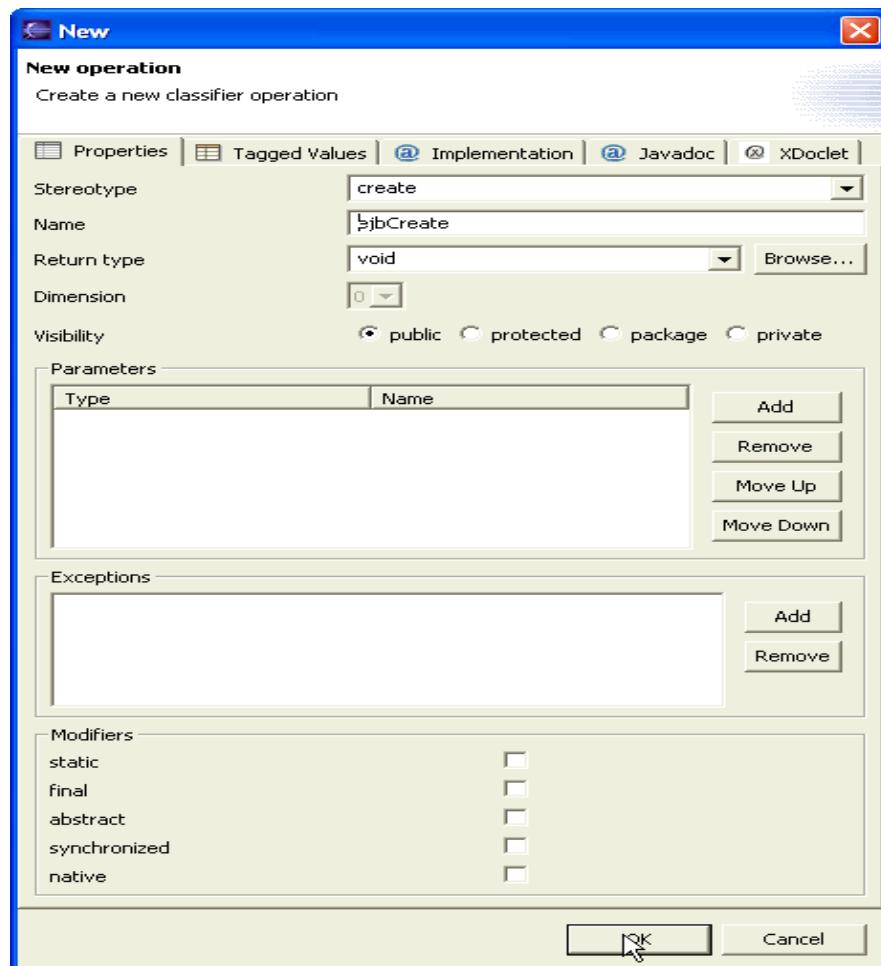


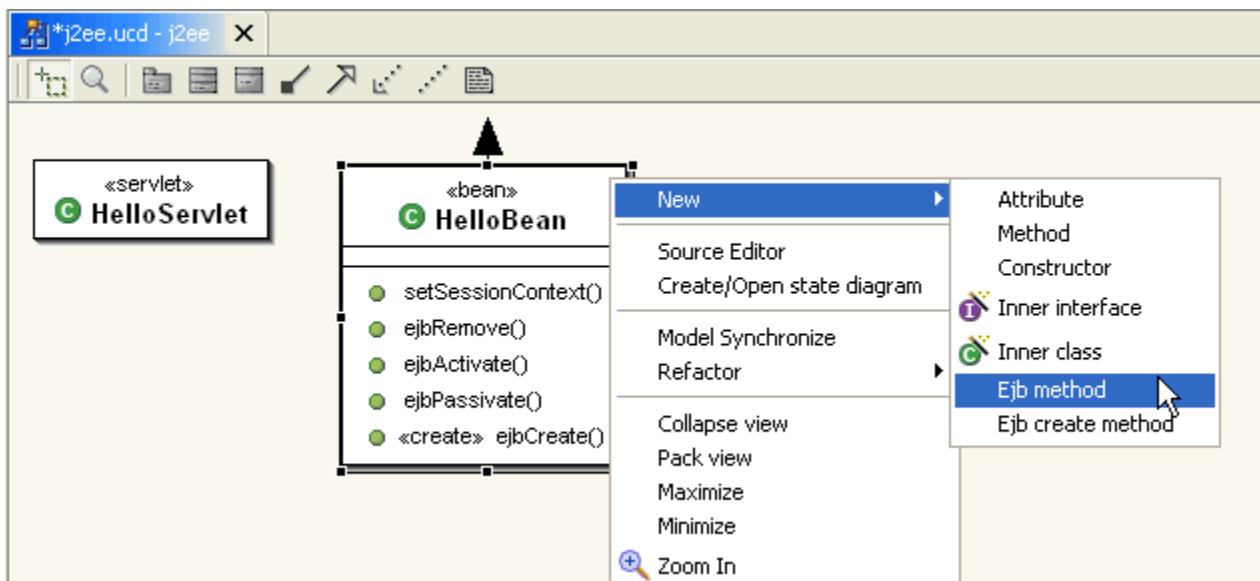
3. Add 'business' and 'create' method to Ejb

If you wish to add a 'create' method. Select a Class with a bean as a stereotype and use the Class Diagram **popup menu** > New > Ejb create method

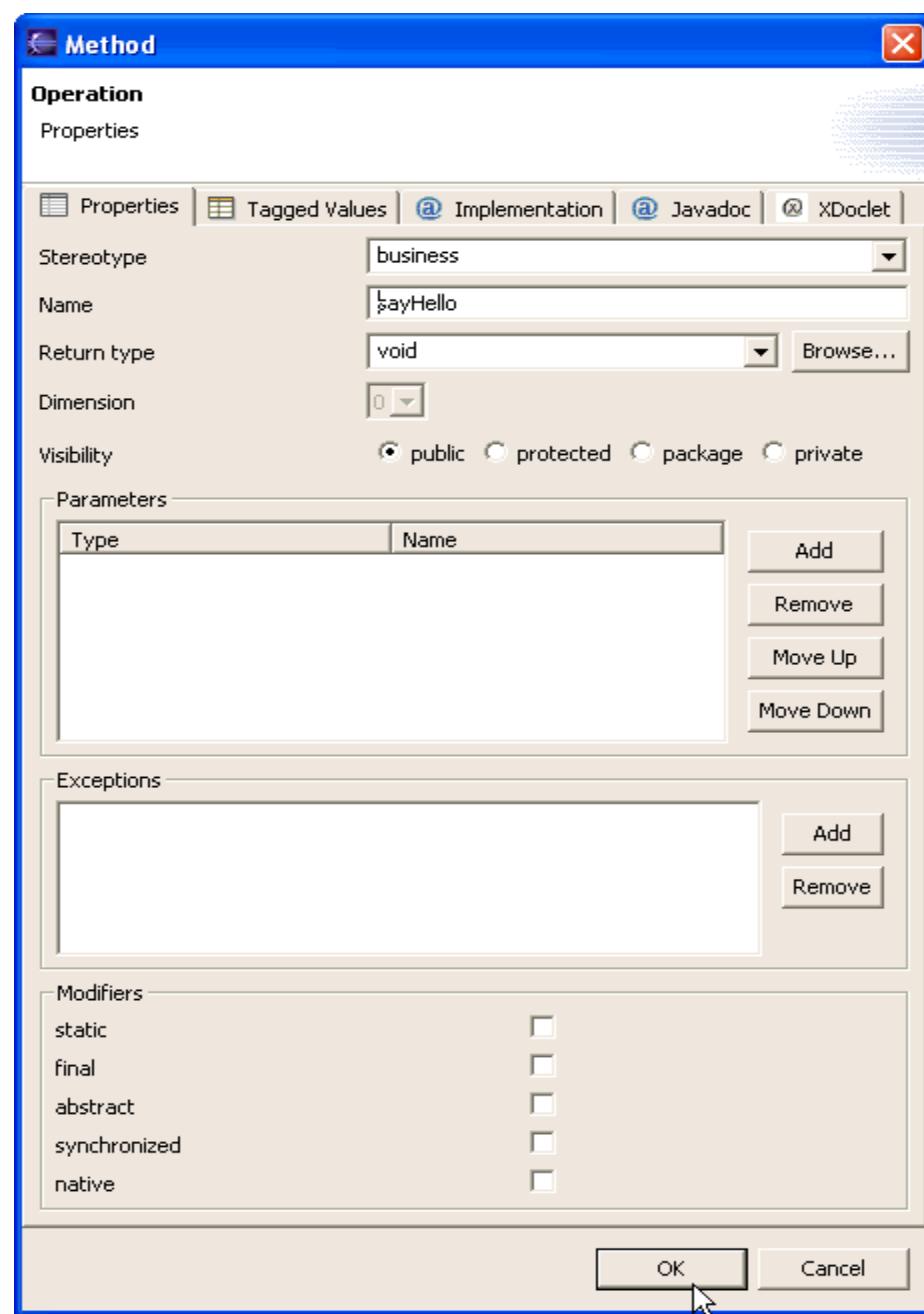


The method stereotype should be "create" in order to have access to the Xdoclet properties, if you change the "create" stereotype, then the Xdoclet tab will disappear.





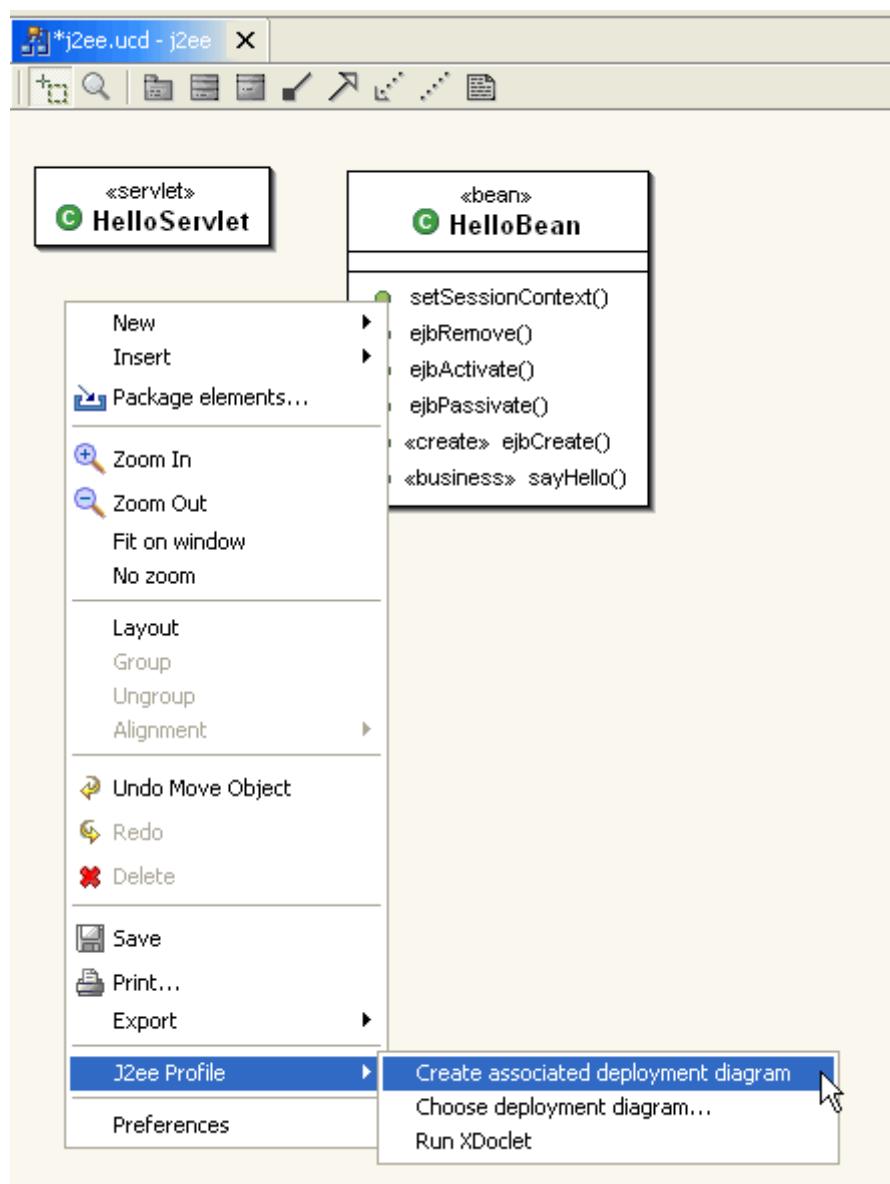
The method stereotype should be "business" in order to have access to the Xdoclet properties, if you change the name of this specific stereotype, then the Xdoclet tab will disappear.



Deployment Configuration

From the Class Diagram you can:

1. [Create an associated deployment diagram](#). This allows you to generate a deployment diagram from a Class Diagram.
2. Choose deployment diagram...This allows you to associate this Class Diagram with an existing Deployment Diagram.
3. Run XDoclet. This allows you to run XDoclet and create the java classes needed for your application.



If you have already created a Deployment Diagram, then the J2ee profiles popup menu changes.



Deployment Diagram creation wizard

From the Class Diagram popup menu, use the **J2ee Profile > Create associated deployment diagram** menu item.

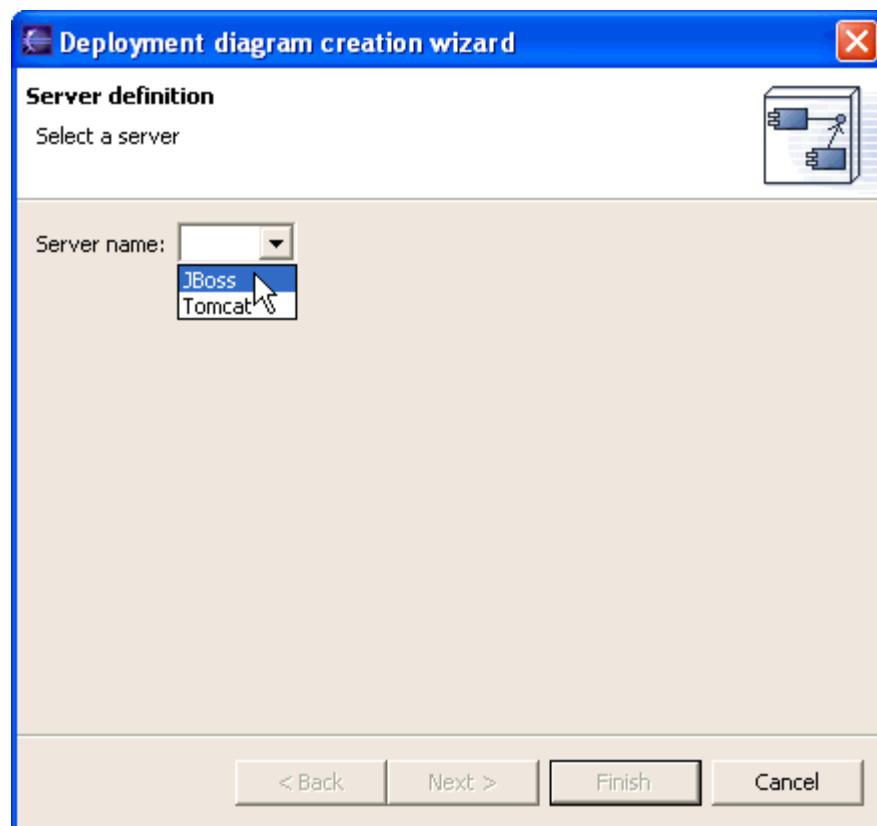
The deployment diagram creation wizard contains four pages:

1. [Server selection to run the modules on.](#)
2. [Enterprise deployment ARchive \(EAR\) definition.](#)
3. [Web modules definition.](#)
4. [Ejb modules definition.](#)

1. Server selection

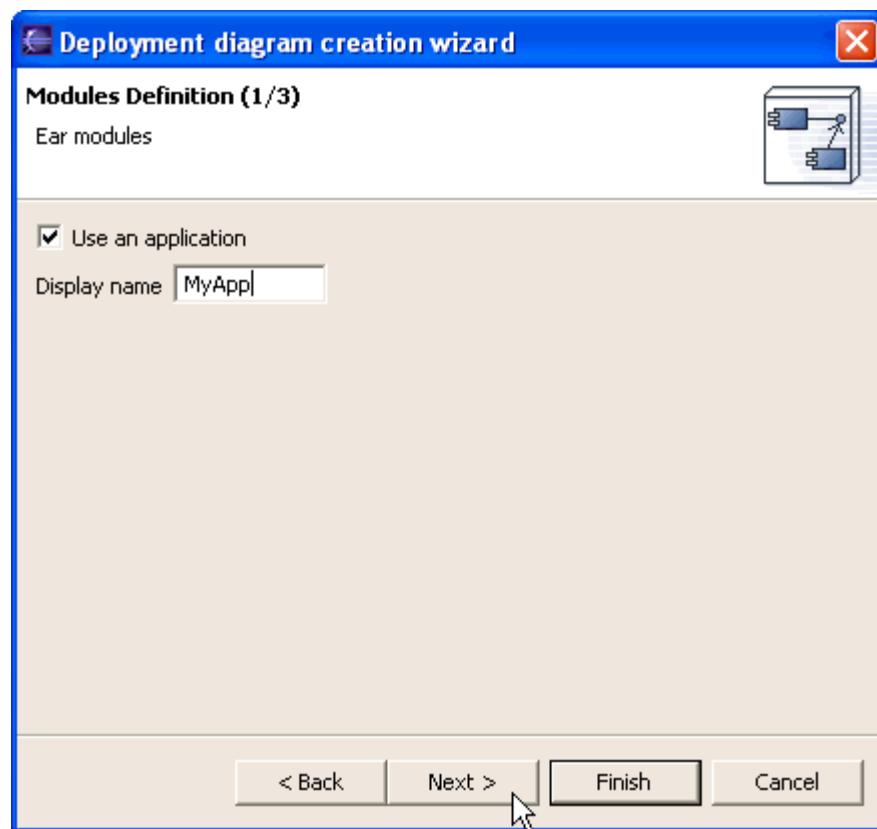
Select the server name by using the drop down arrow.

Select JBoss application server.



2. Enterprise deployment ARchive (EAR) definition.

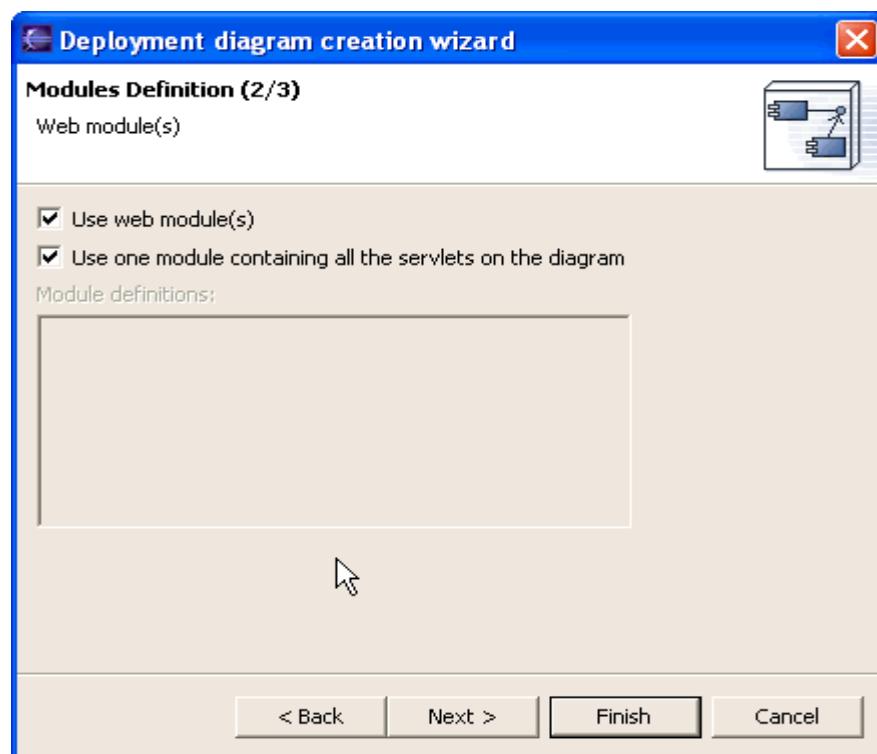
You can choose to create an application to include web and ejb modules.



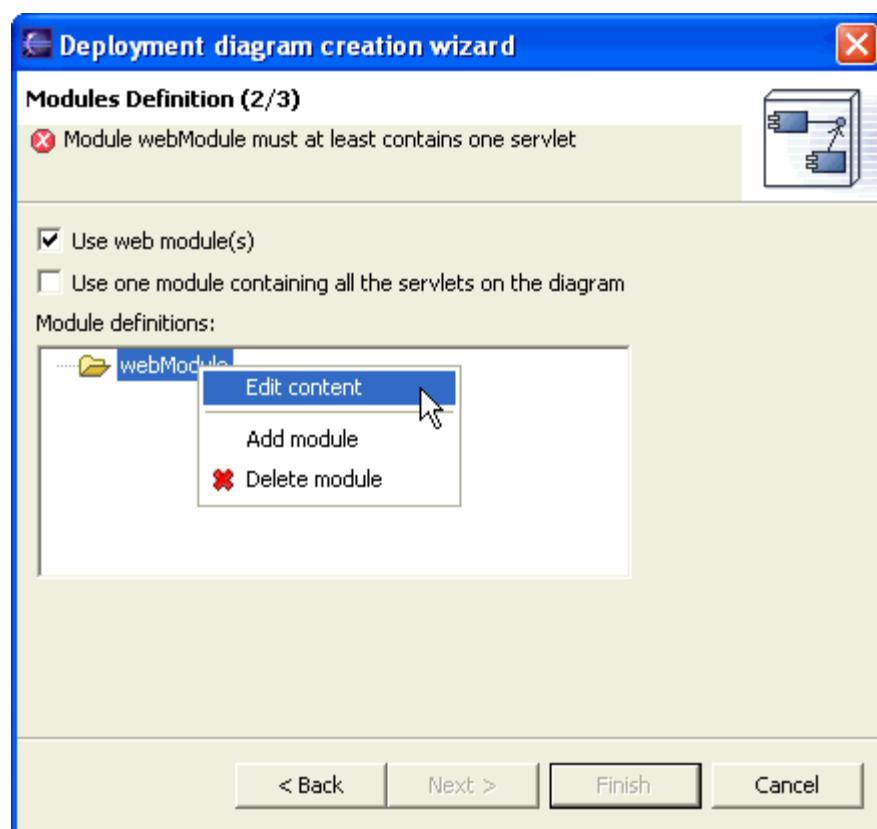
3. Web ARchive (WAR) definition

You can choose to use web modules.

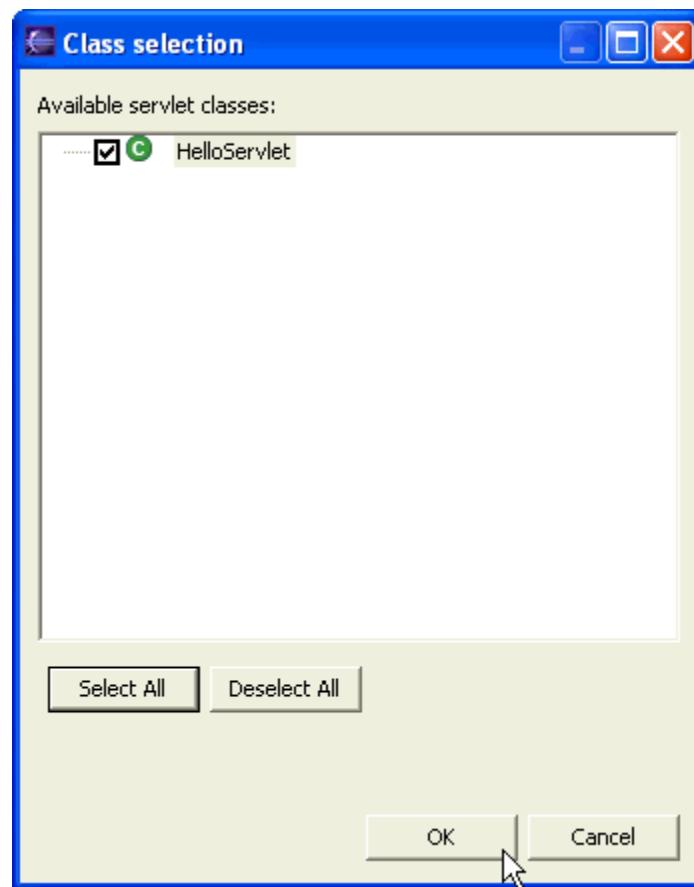
You can either use one module containing all the servlets which are presently located on the diagram or define modules manually.



Uncheck the second checkbox to enable the module editor.
Use the popup menu to add a new module.
Click on webmodule to open the popup menu and **select edit content**

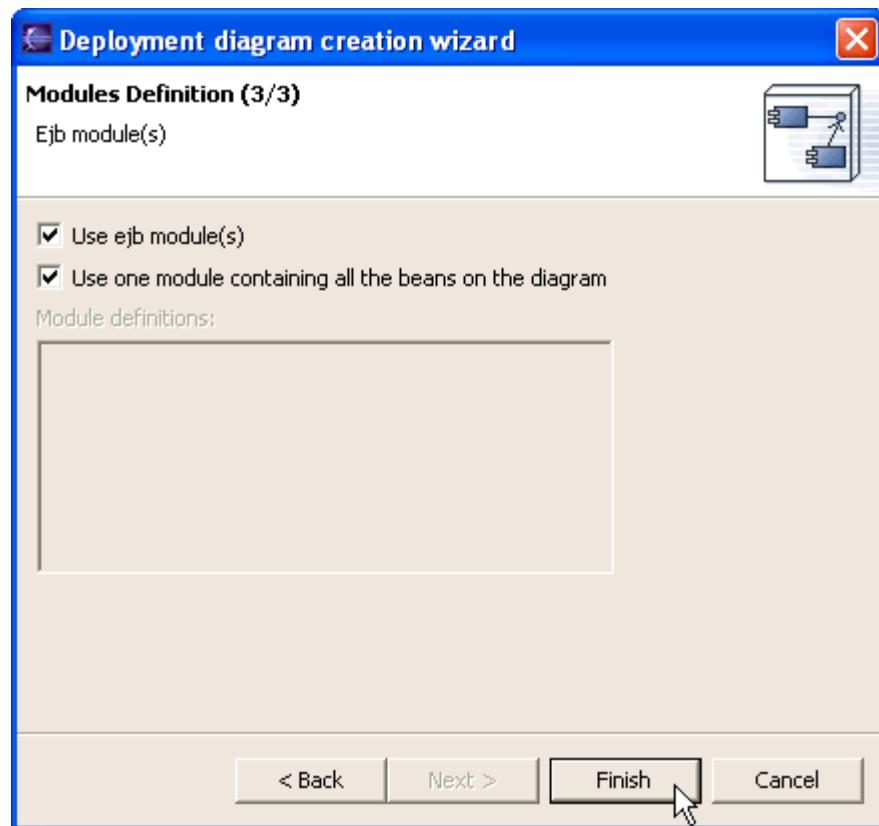


This action opens the following dialog which contains all the servlets of the current project:

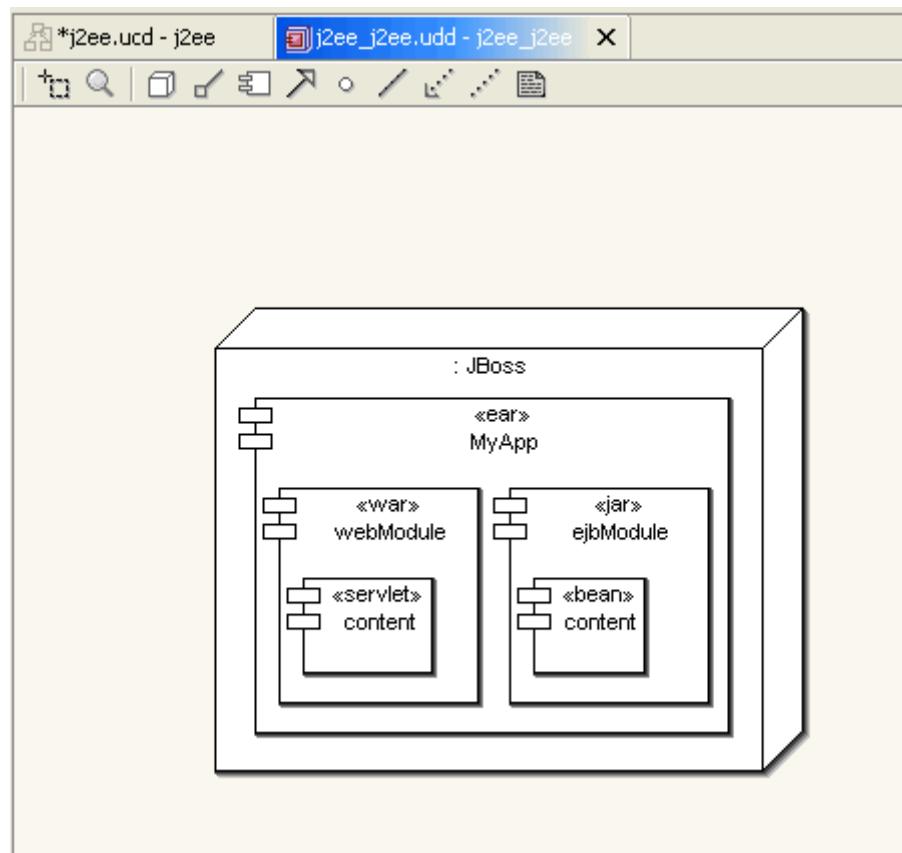


4. Ejb modules definition

This page works in the same way as the web module definition page.



Clicking on the finish button will generate a deployment diagram which should look like the following image:



Managing a Server

Please note that you can create a J2ee deployment diagram from scratch or use a [wizard](#) from a class diagram.

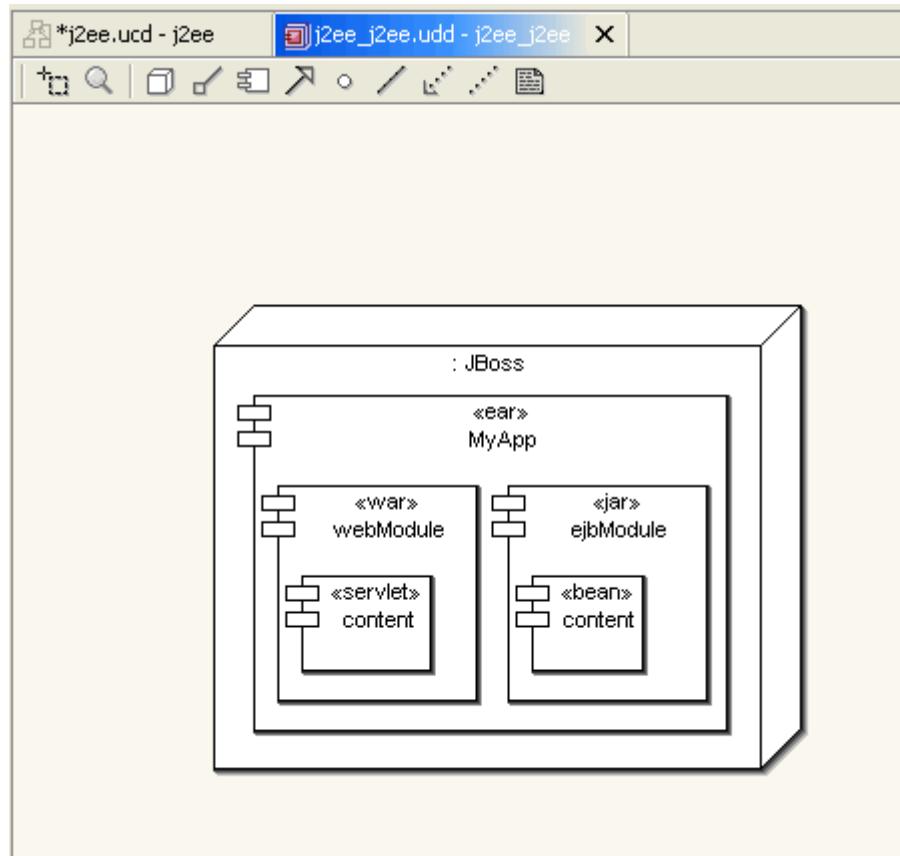
If you want to set up a deployment diagram manually, you have to make sure you set the right stereotypes to the components.

- A component with the 'ear' stereotype represents an Application.
- A component with the 'war' stereotype represents a Web Module.
- A component with the 'jar' stereotype represents an Ejb Module.
- A component with the 'servlet' stereotype represents a list of servlets
- A component with the 'bean' stereotype represents a list of ejb.

This section covers the following elements:

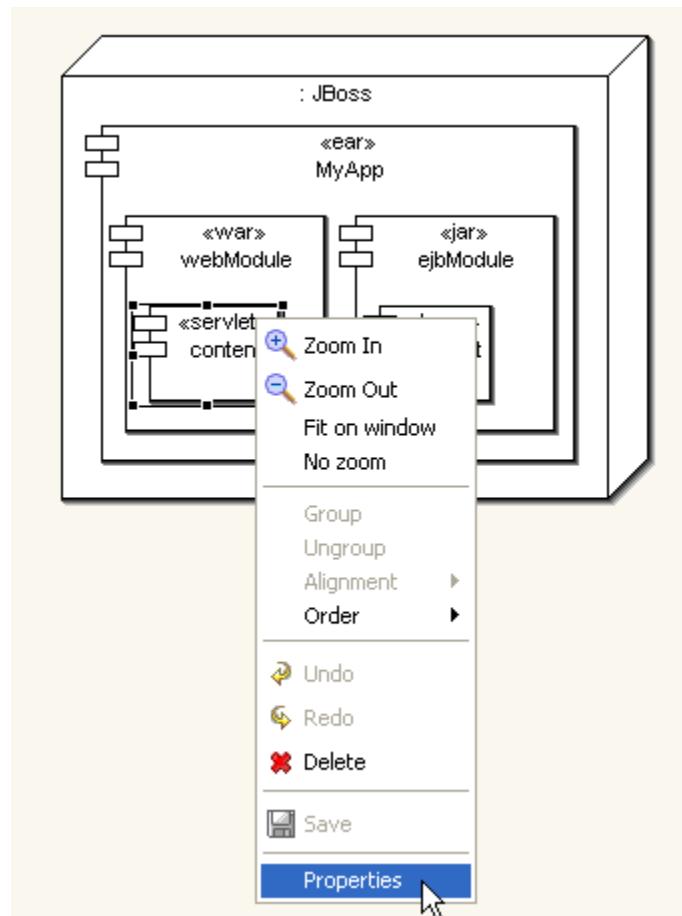
1. [Web Module properties](#)
2. [Ejb Module properties](#)
3. [Application properties](#)
4. [Nodes properties](#)
5. [Deploying modules/application](#)
6. [Managing a server](#)

The following Deployment Diagram has been created from a Class Diagram by a [wizard](#).

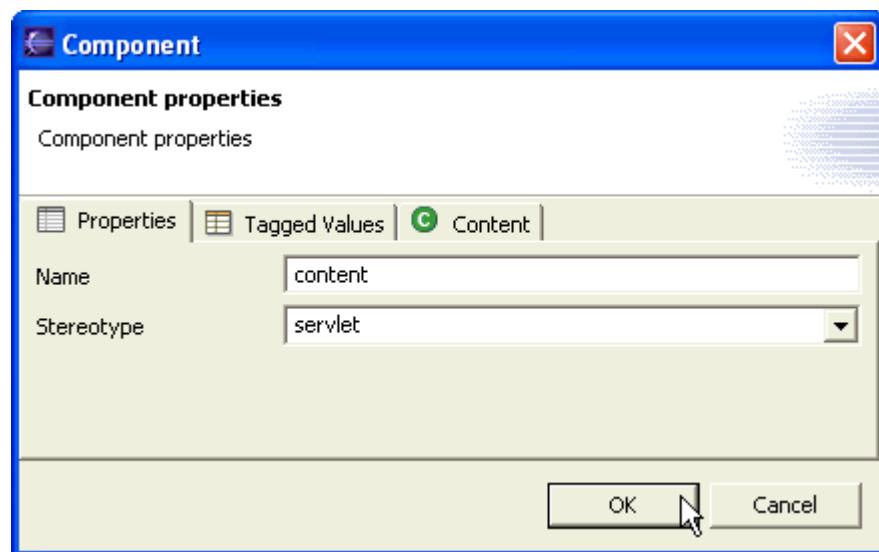


1. Web Module properties

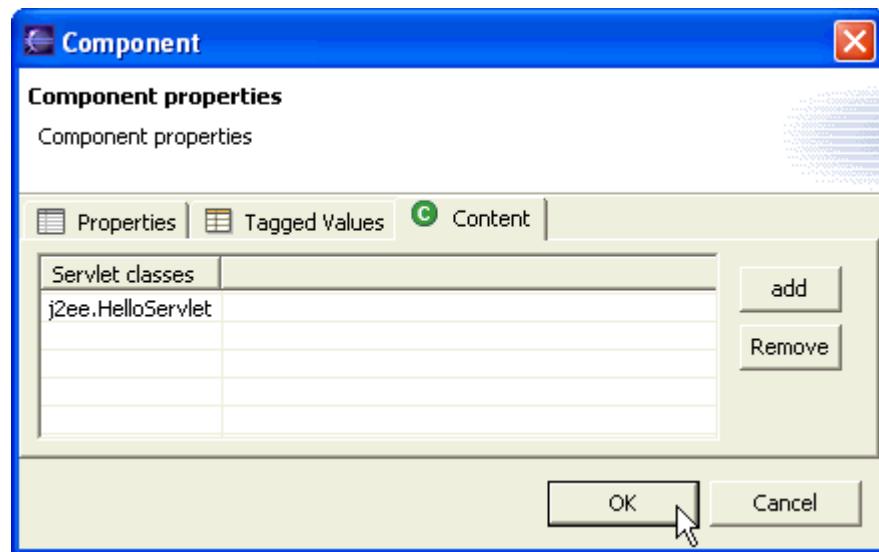
Select the 'servlet' component inside the JBoss node and click on it.



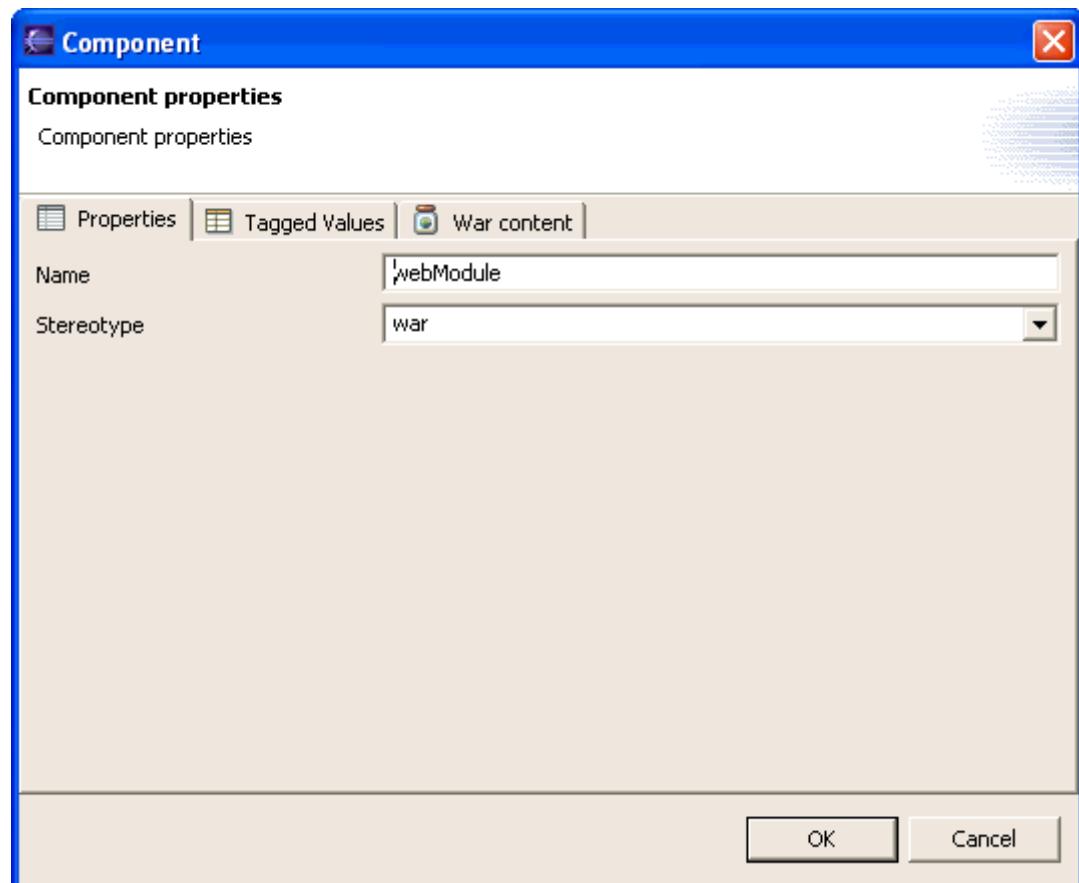
O
Open the component properties **popup menu >properties**



Enter the content tab, which is specific to 'servlet' components.
We can change the servlet list held by this component.

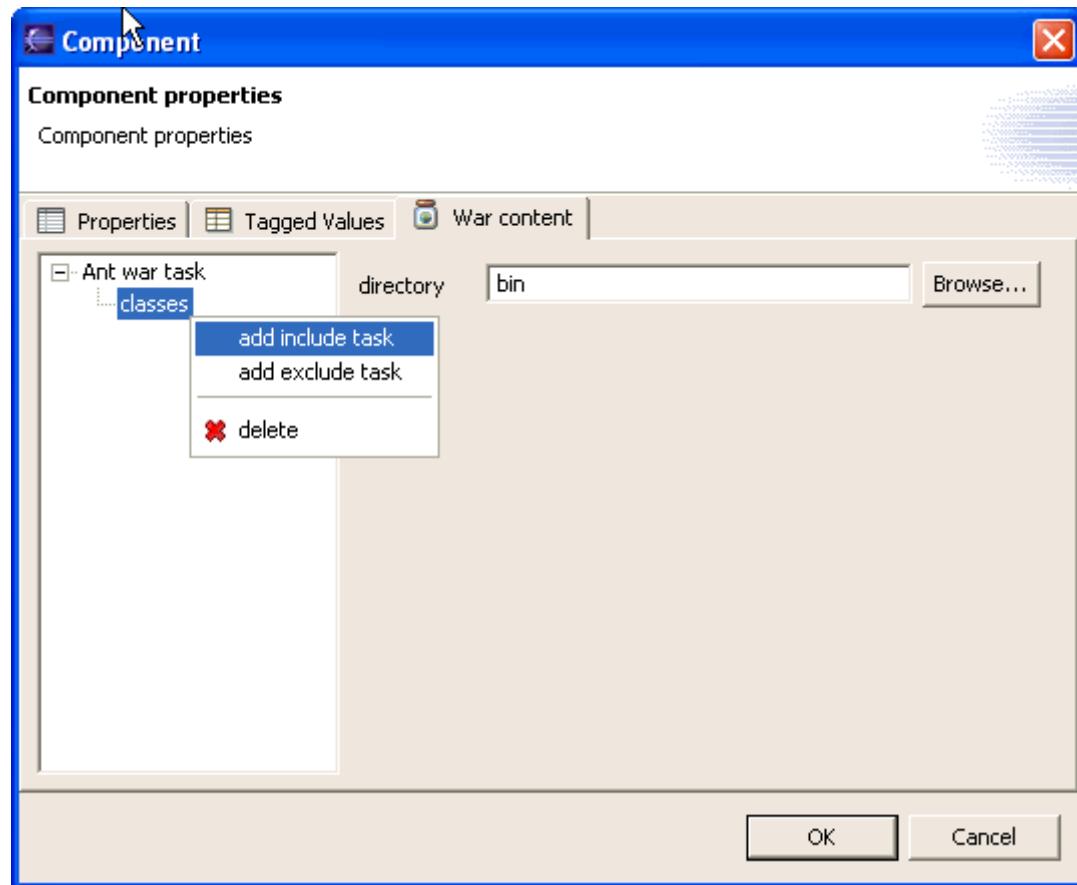


Select the web module component inside the JBoss node and click on it.
Open **the web module popup menu >properties**

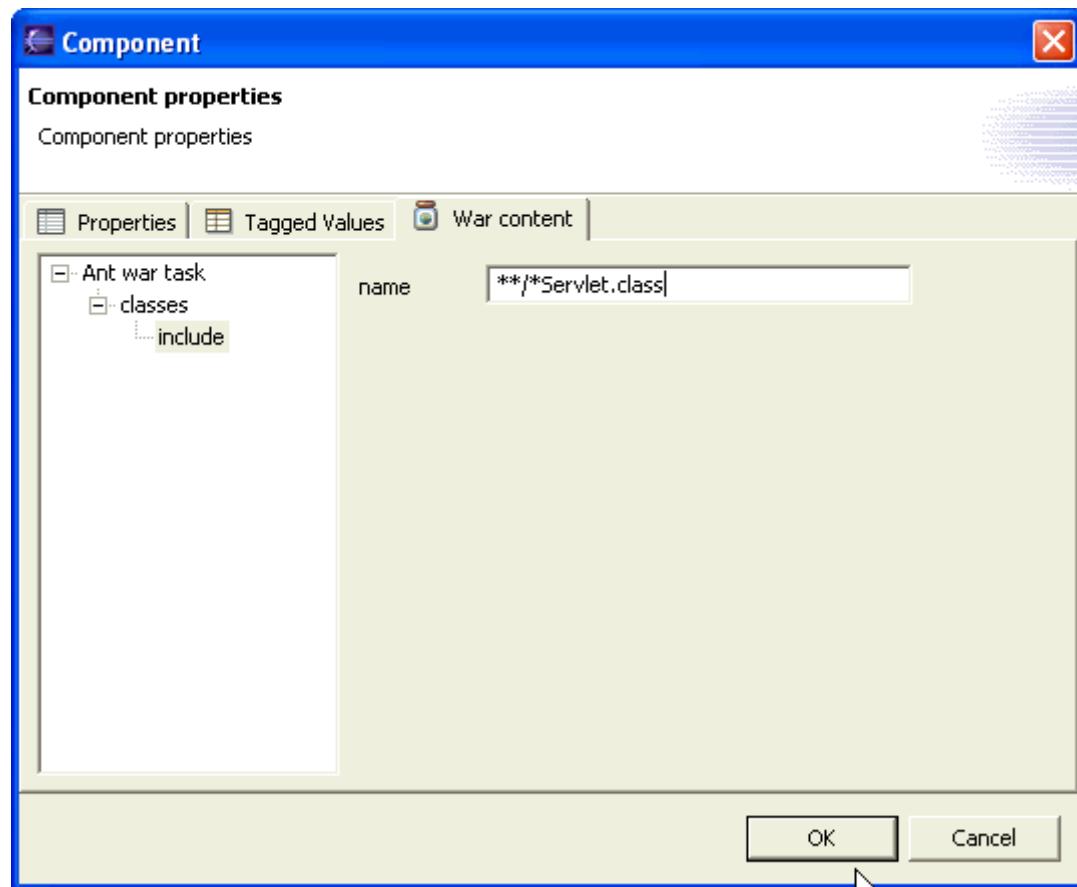


Enter the War content tab, which is specific to 'war' components.

Select the Ant war task and select the content of the war file that will be generated at the deployment time.

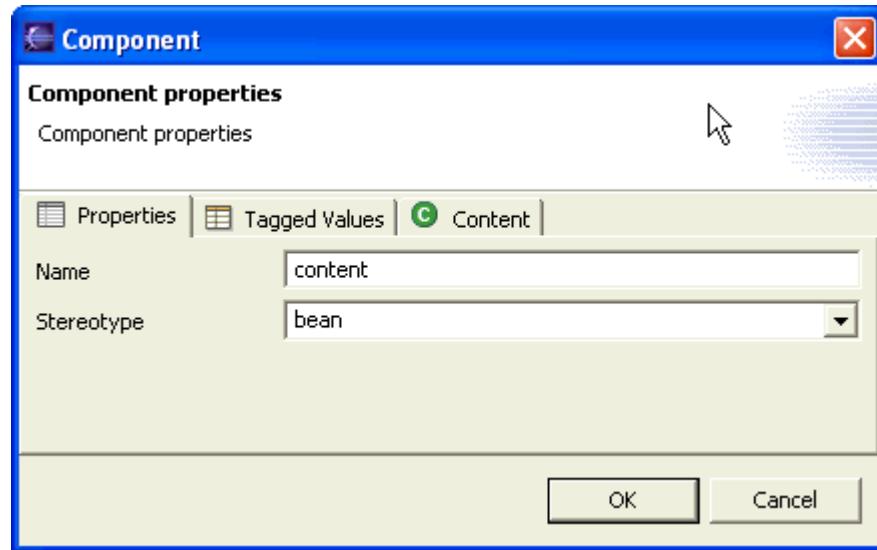


We can add a filter or keep the default value.

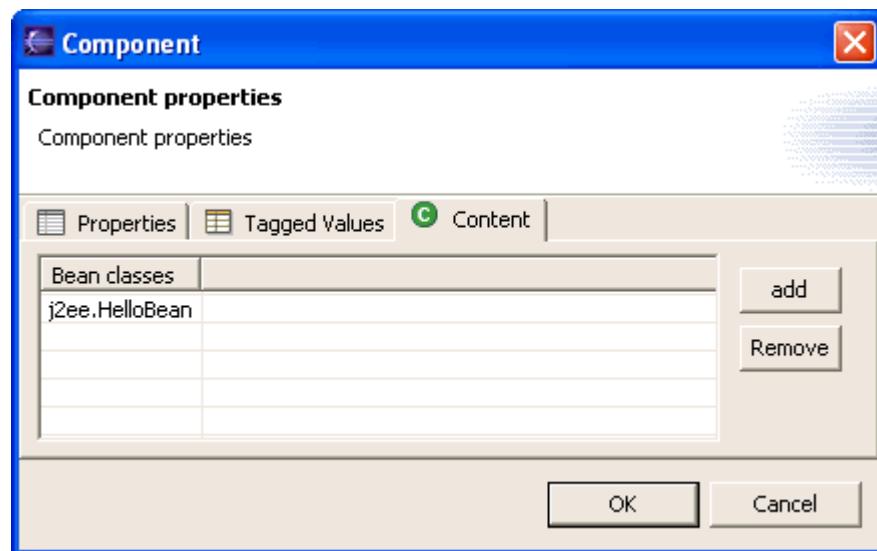


2. Ejb Module properties

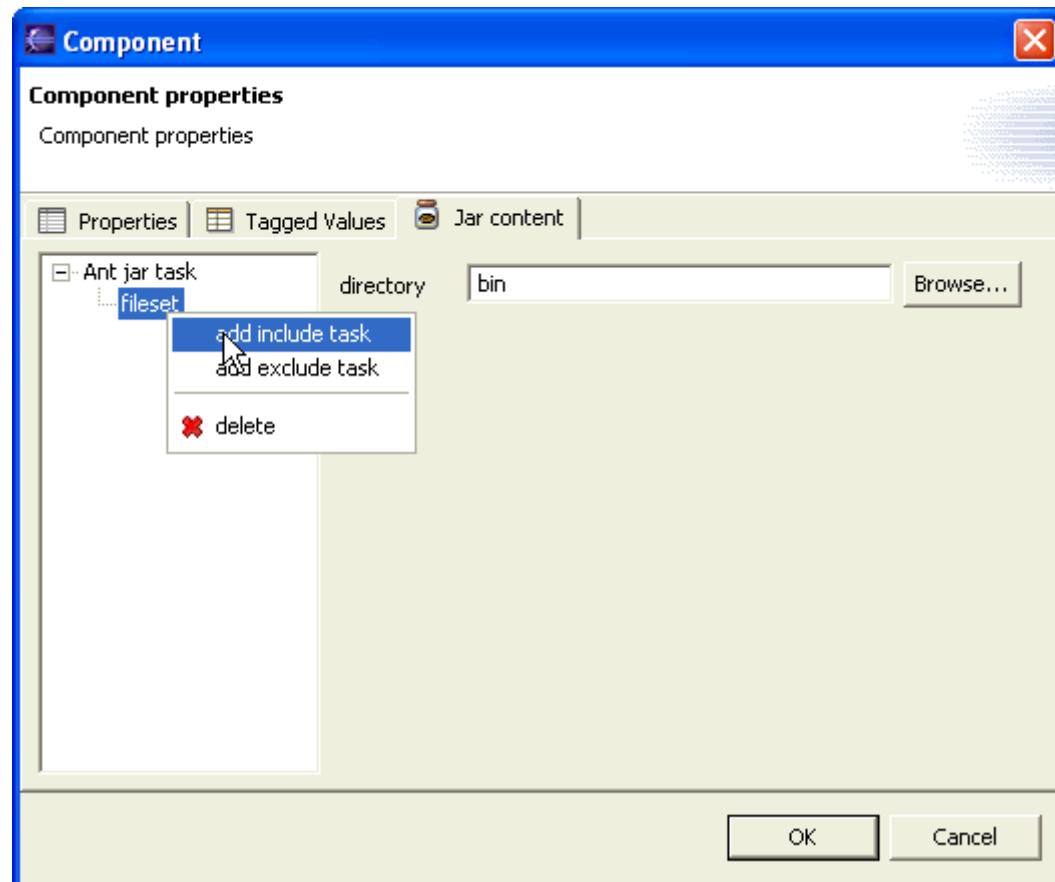
Select the 'bean' component inside the JBoss node and click on it.
 Open the **Bean** popup menu >properties



Enter the content tab, which is specific to 'bean' components.
 We can change the bean list held by this component.

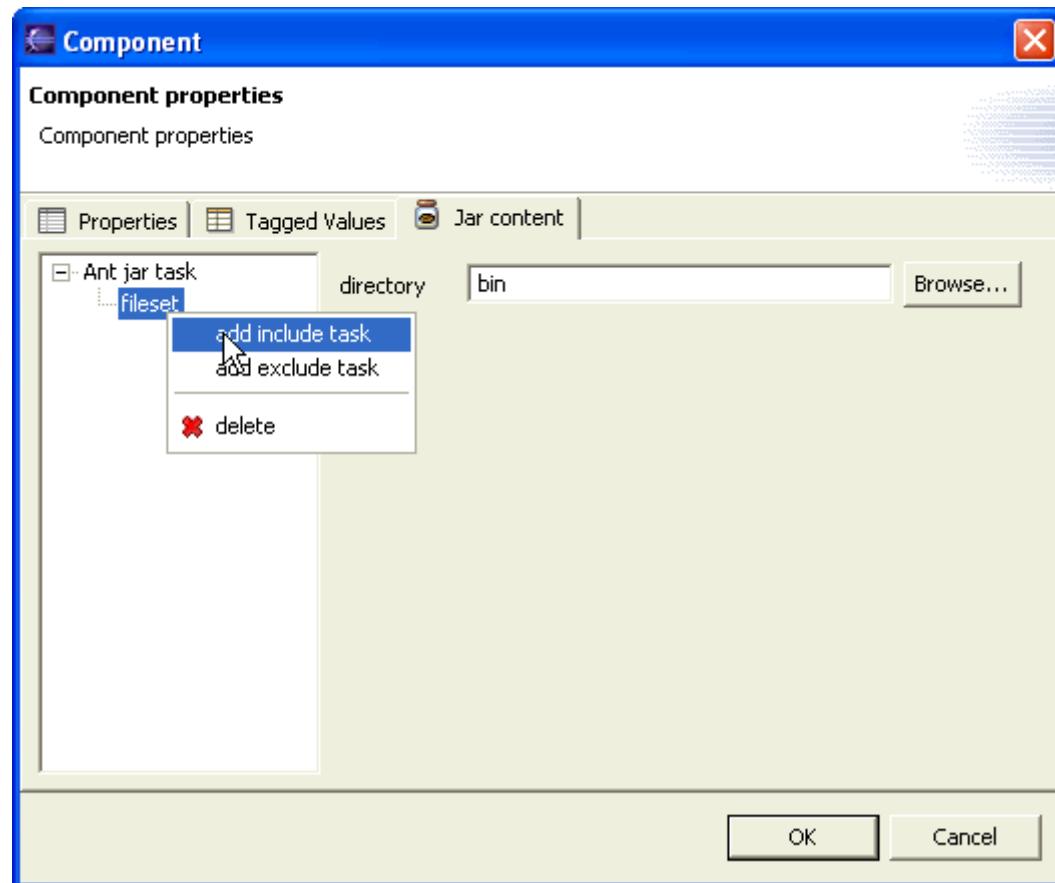


Select the Ejb module component inside the JBoss node and click on it.
 Open the **Ejb** module popup menu >properties

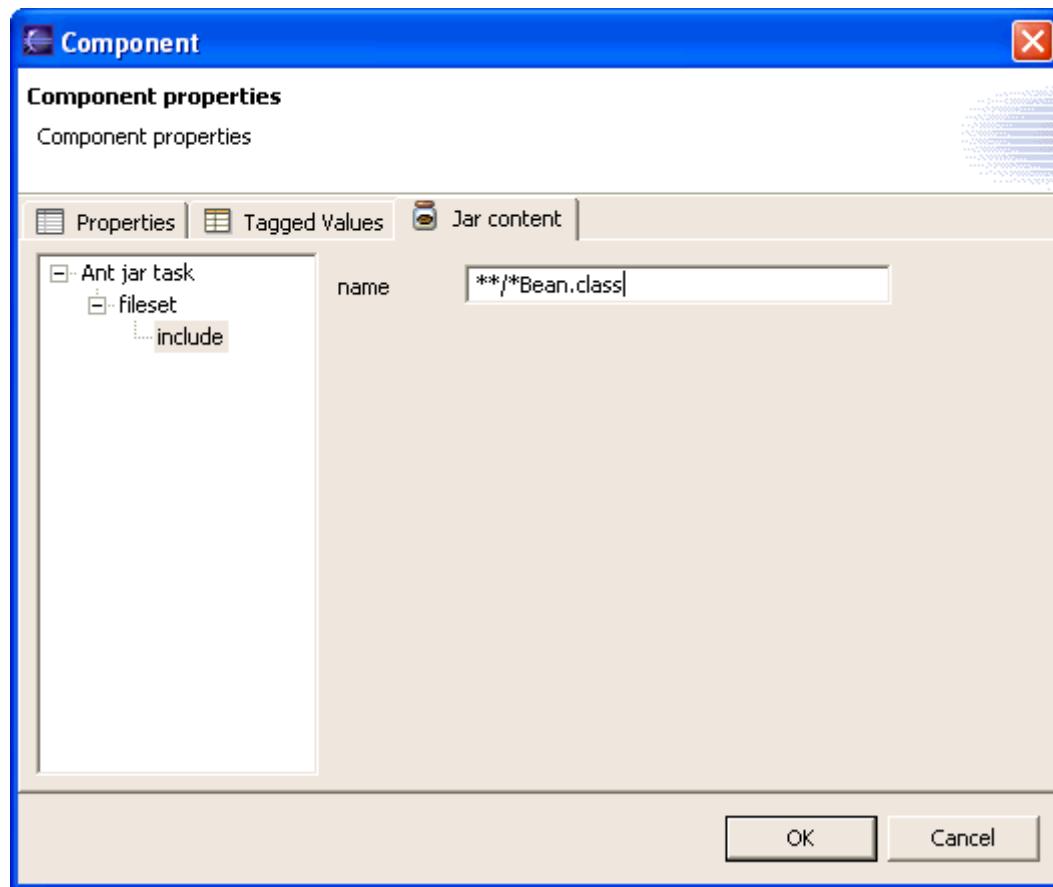


Enter the Jar content tab, which is specific to 'ejb' components.

Select the Ant jar task and select the content of the jar file that will be generated at the deployment time.

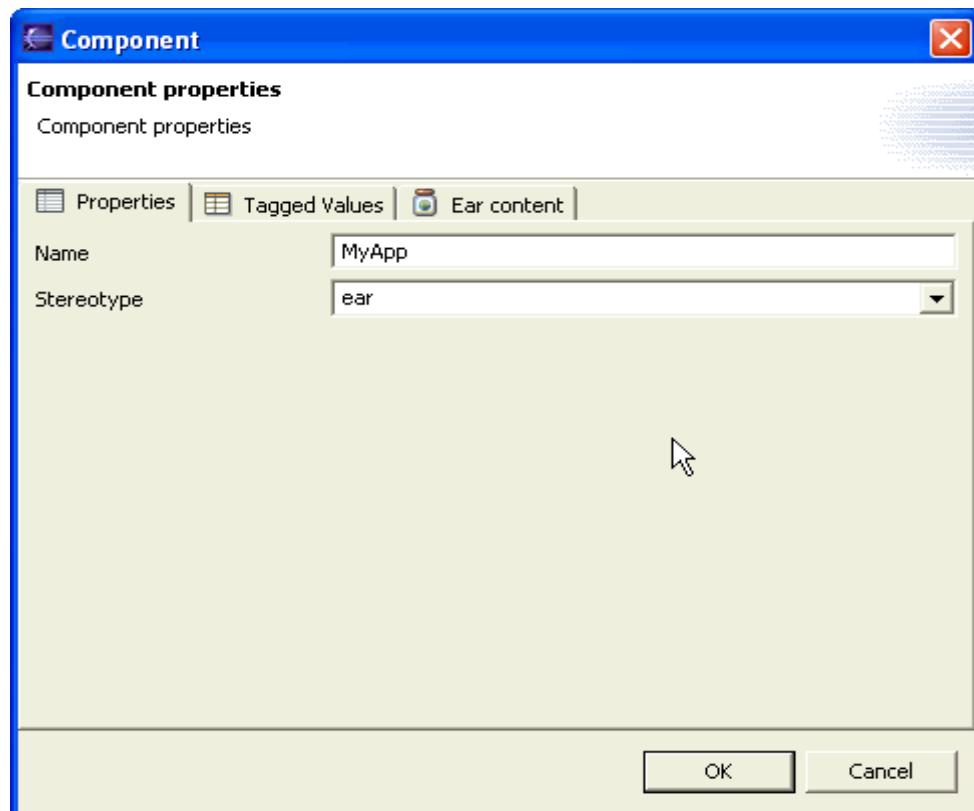


We can add a filter or keep the default value.

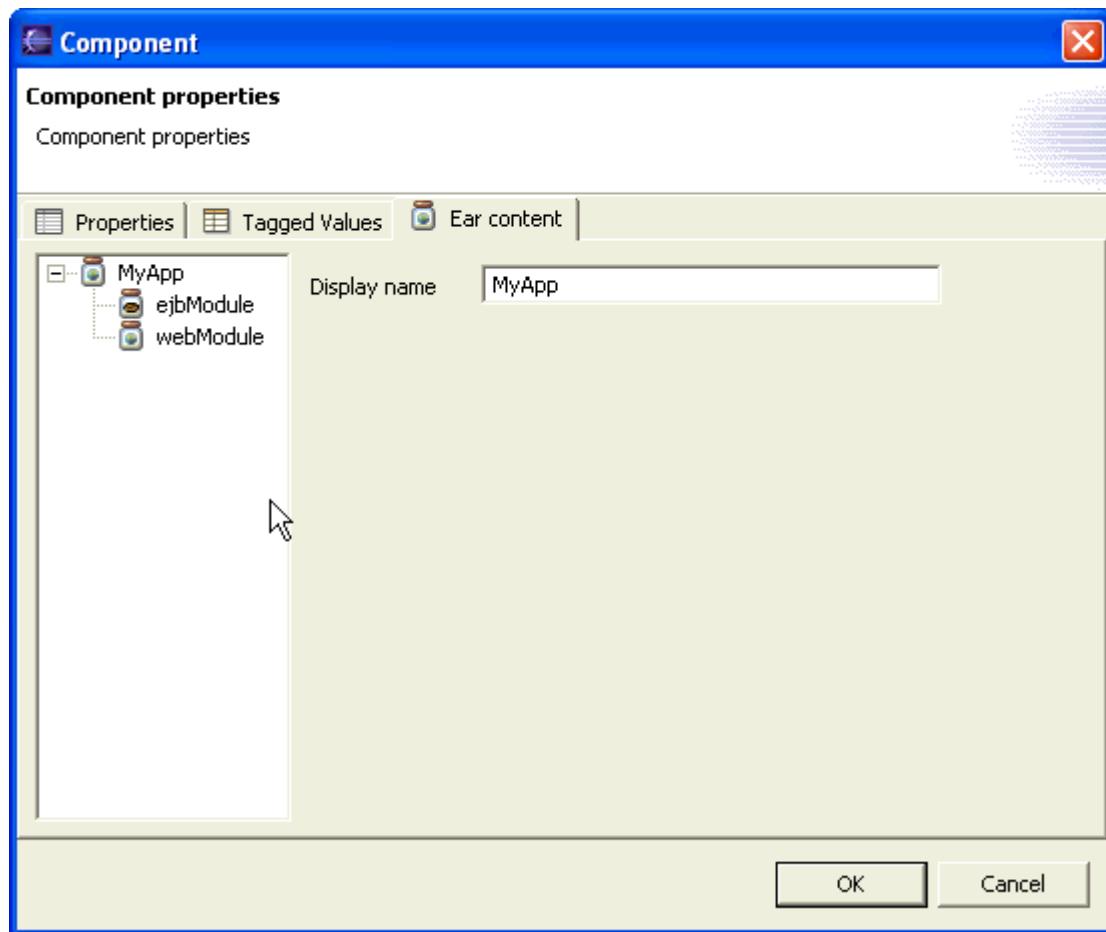


3. Application Module properties

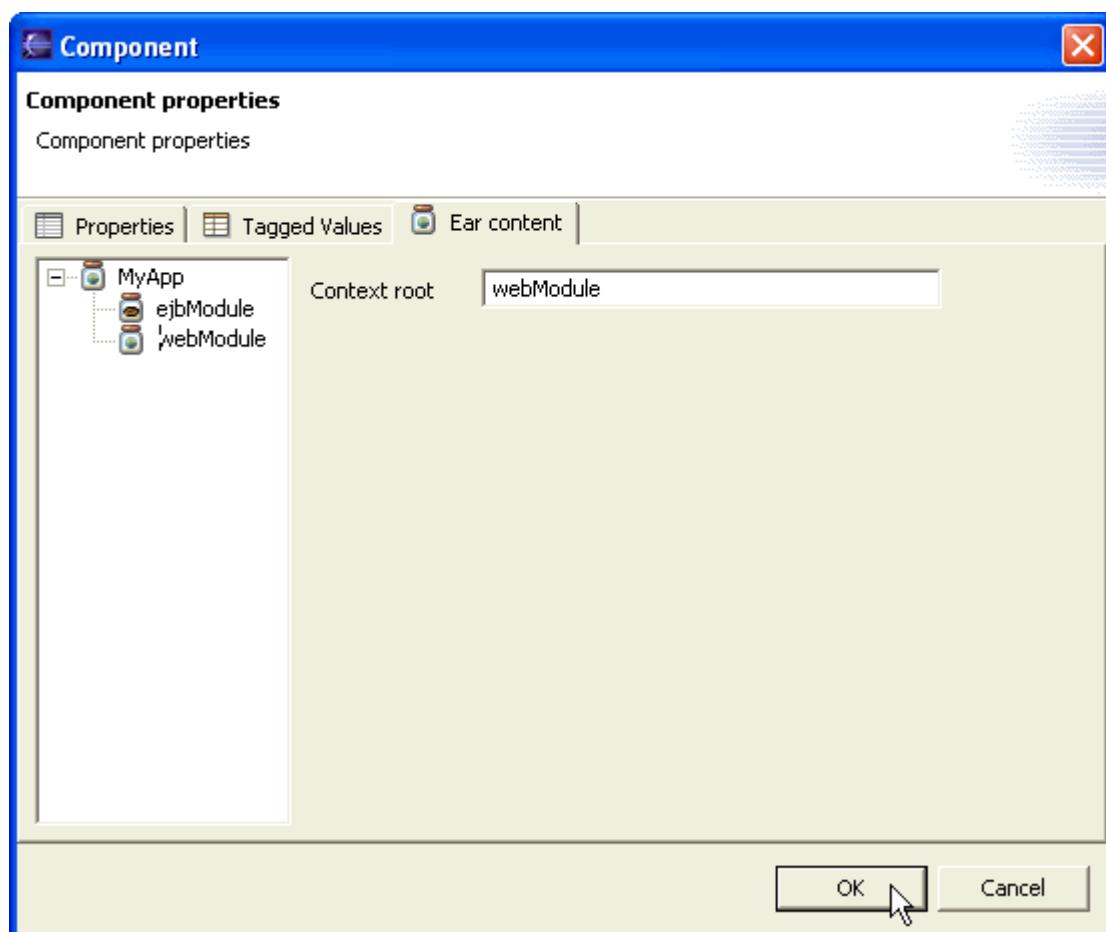
Select the MyApp component inside the JBoss node and click on it.
Open the MyApp **component popup menu >properties**



Enter the Ear content tab, which is specific to 'ear' components.
In the left pane we can see all modules which are used inside this application.



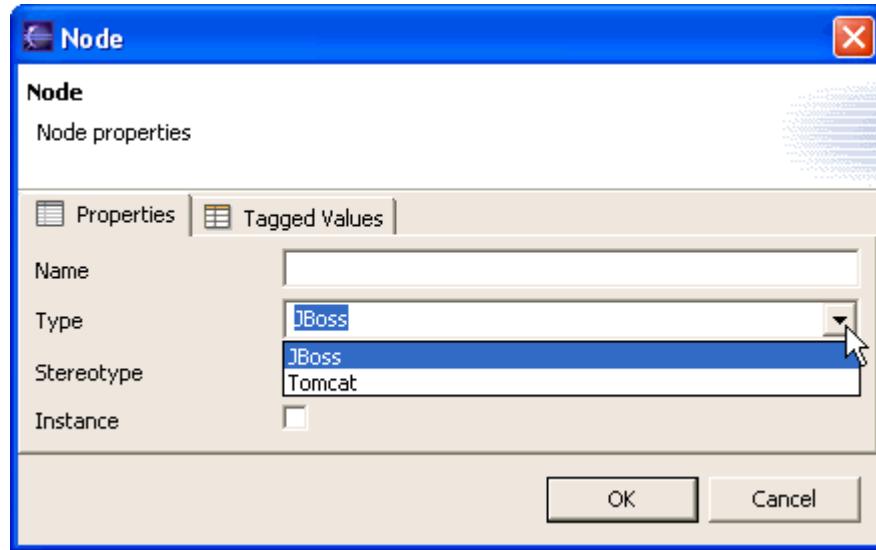
We can change the Context root names of web modules.



4. Node properties

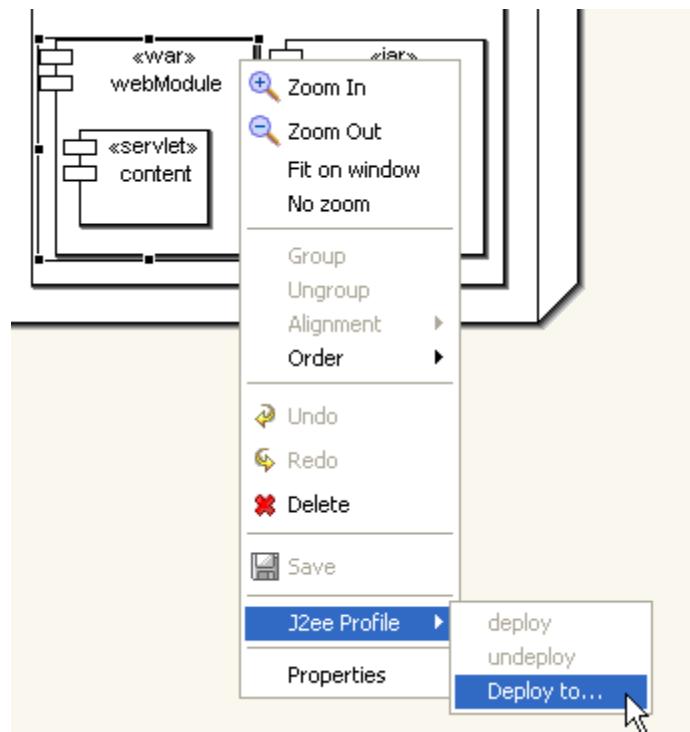
Select the JBoss node and click on it.

Open the JBoss node **popup menu >properties**



5. Deploying modules/application

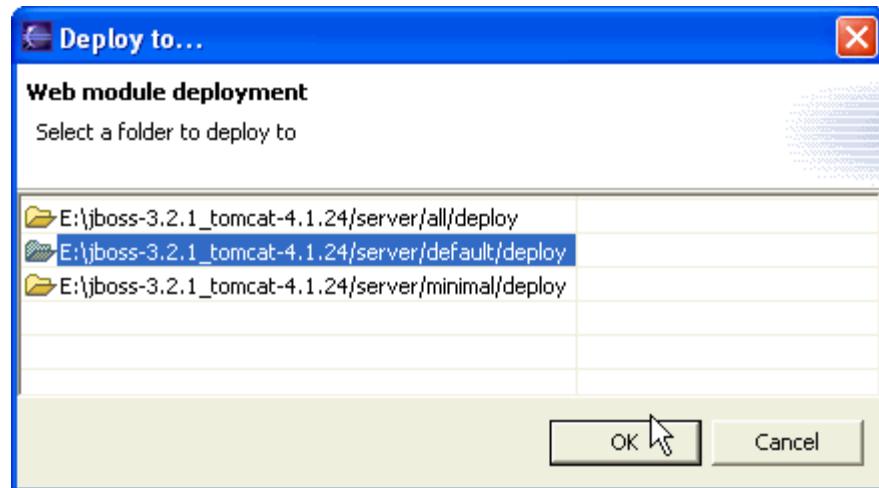
On each modules and application we can manage the deployment of the component by the mean of its context menu.



The menu group contains three items:

- deploy (or redeploy if it is not the first time deployed)
- undeploy
- deploy to (to choose or change the deployment location)

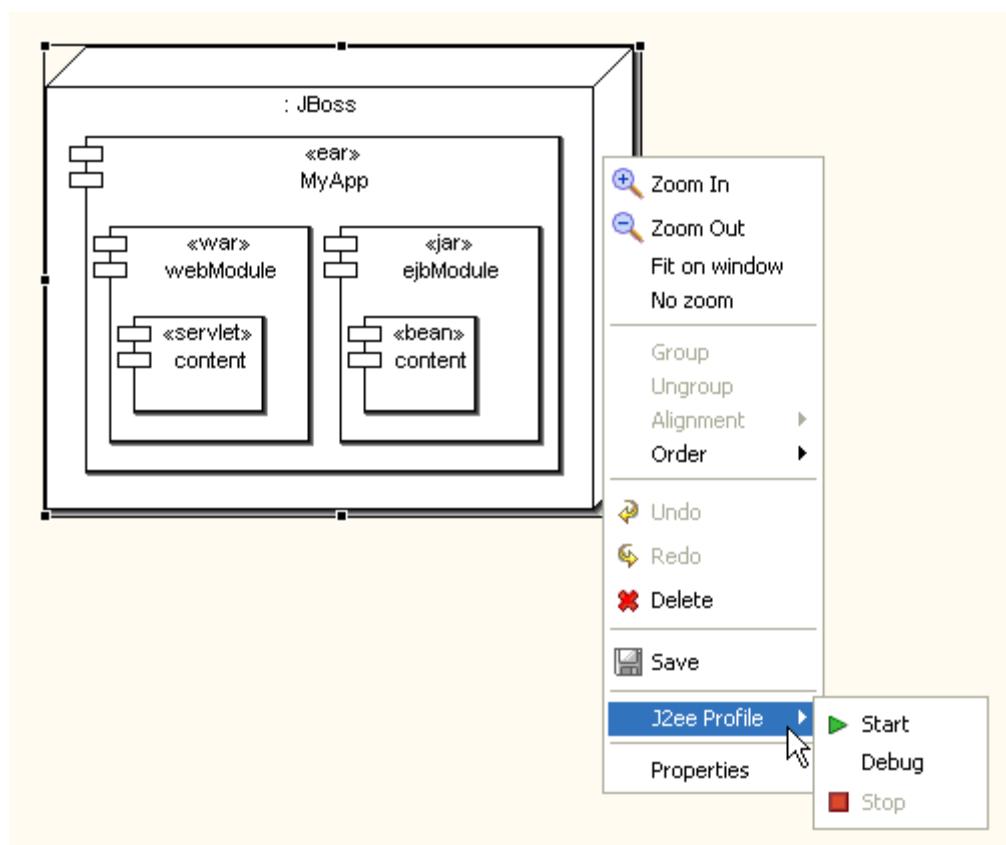
The 'deploy to' menu item will open the following dialog which contains the deployment folders set in the [Server configuration](#).



6. Managing a server

By selecting the JBoss node we can:

- Start
- Debug
- Stop



Server Configuration

Currently two servers are preconfigured, you only need to set up the server path to make them usable for the deployment diagrams.

The next release will allow you to create your own configuration for any j2ee server.

Each J2ee configuration contains the following tabs:

1. [Properties](#)
2. [Start](#)
3. [Stop](#)
4. [Deployment](#)

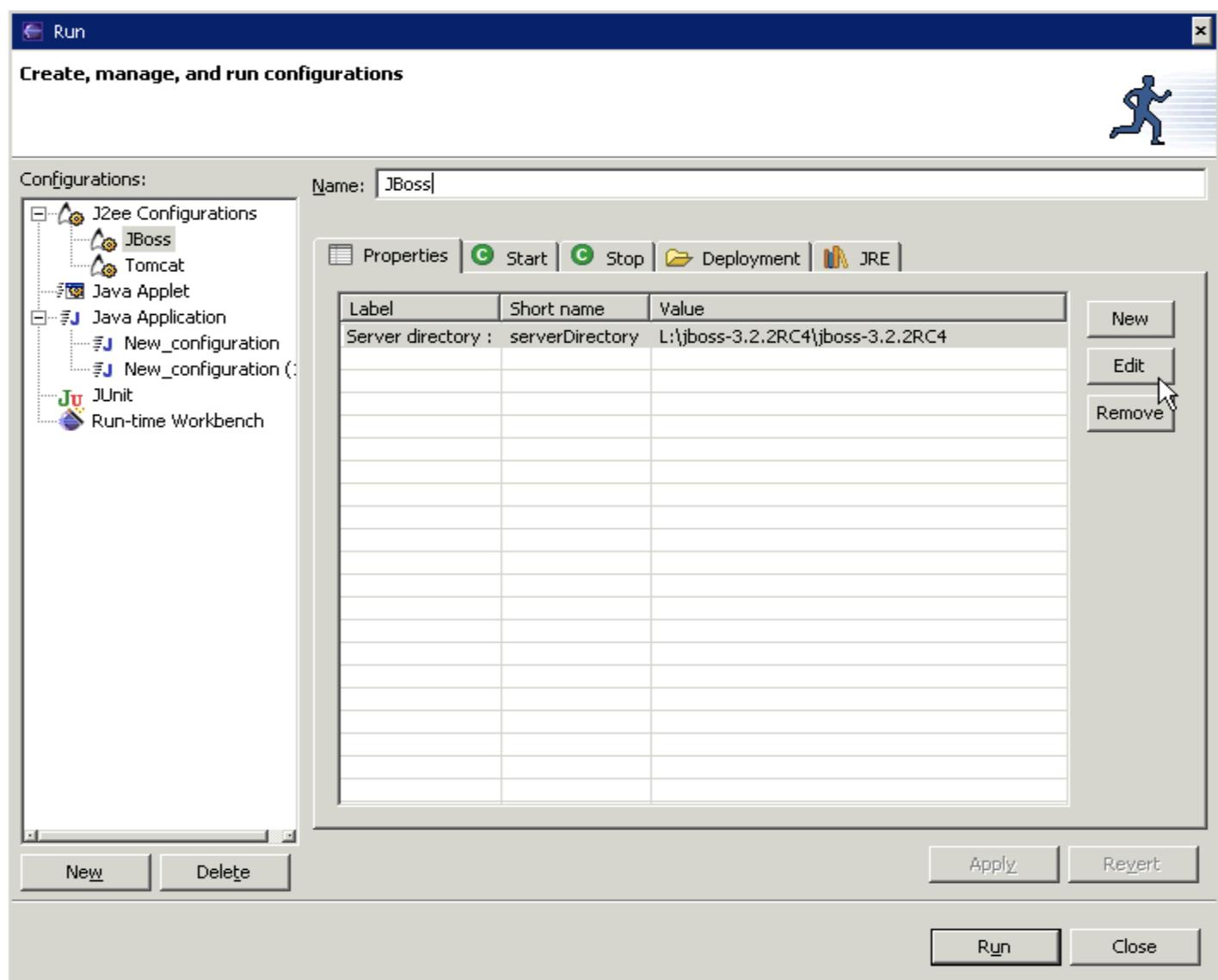
1. Properties tab

You can choose the server directory.

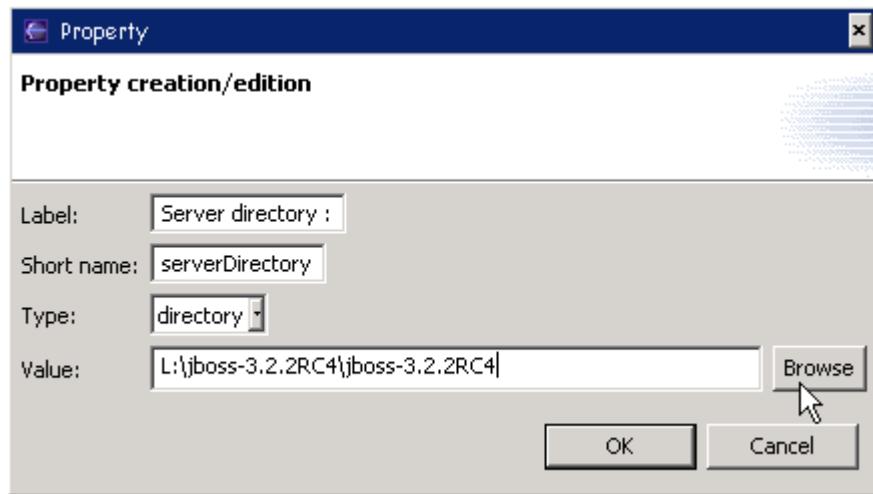
The next release will allow you to add custom properties.

We are going to select JBoss 3.2.2 as an example.

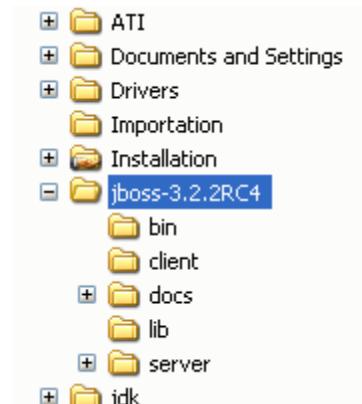
Select **J2ee Configurations > JBoss**.



Tomcat and JBoss servers are built in, to use them just set the right folder. You can choose the server directory. Enter Property values.

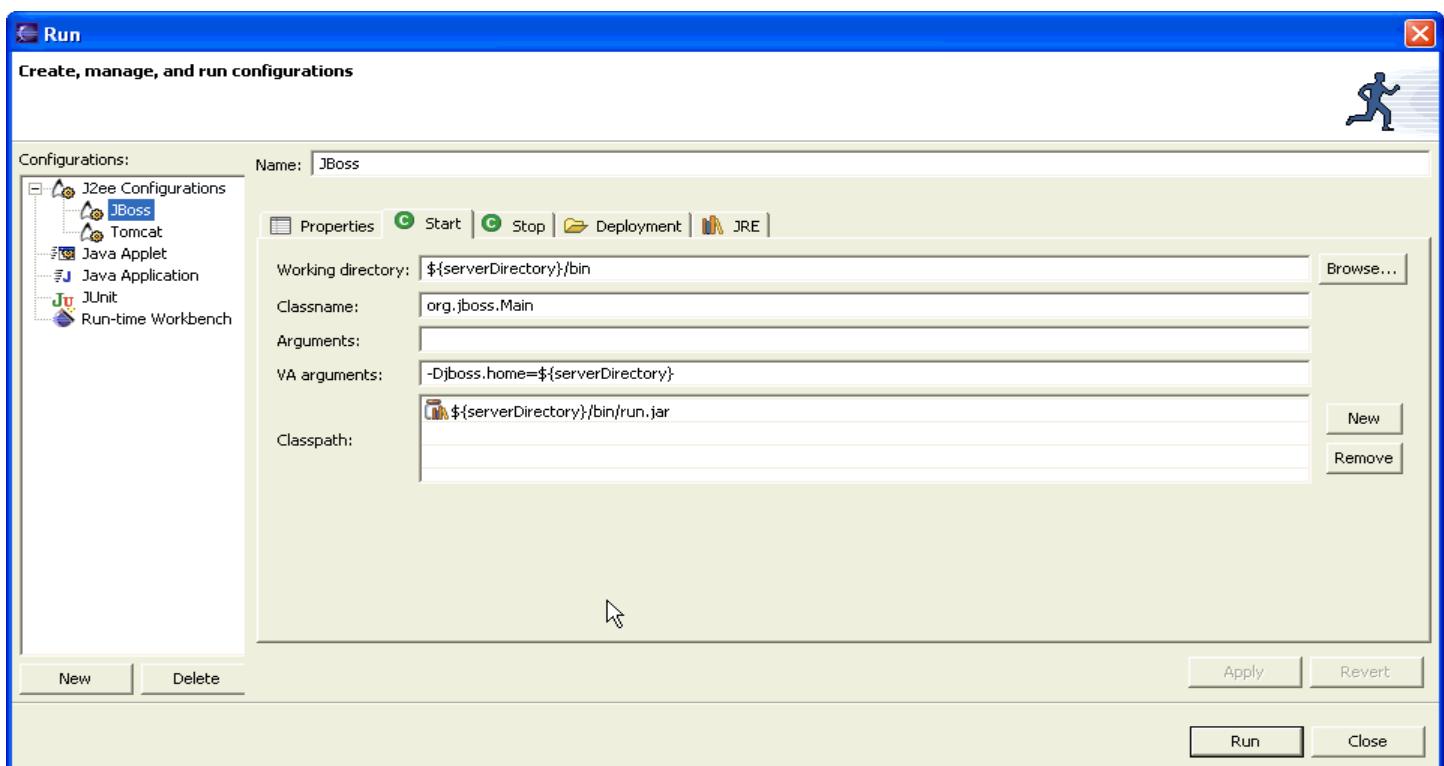


Browse your folders and select jboss-3.2.2RC4.



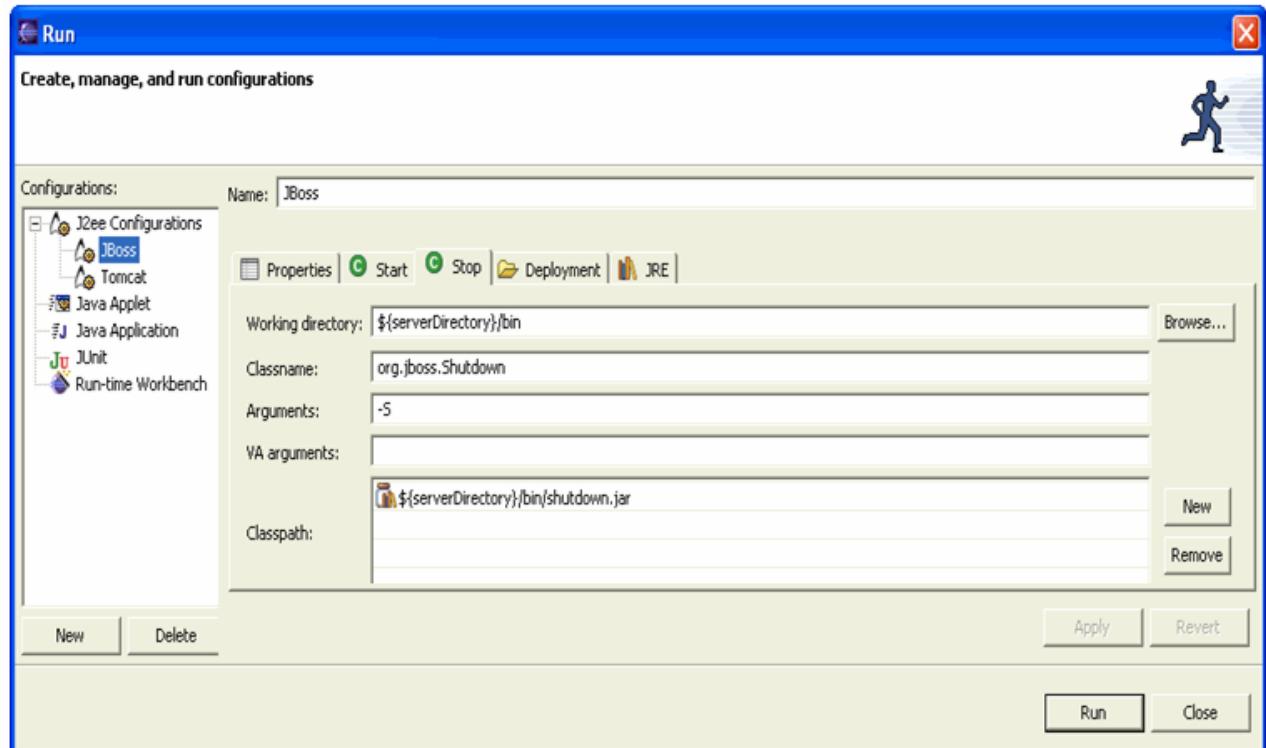
2. Start tab

You can customize the options to start up the server. Default values are available.



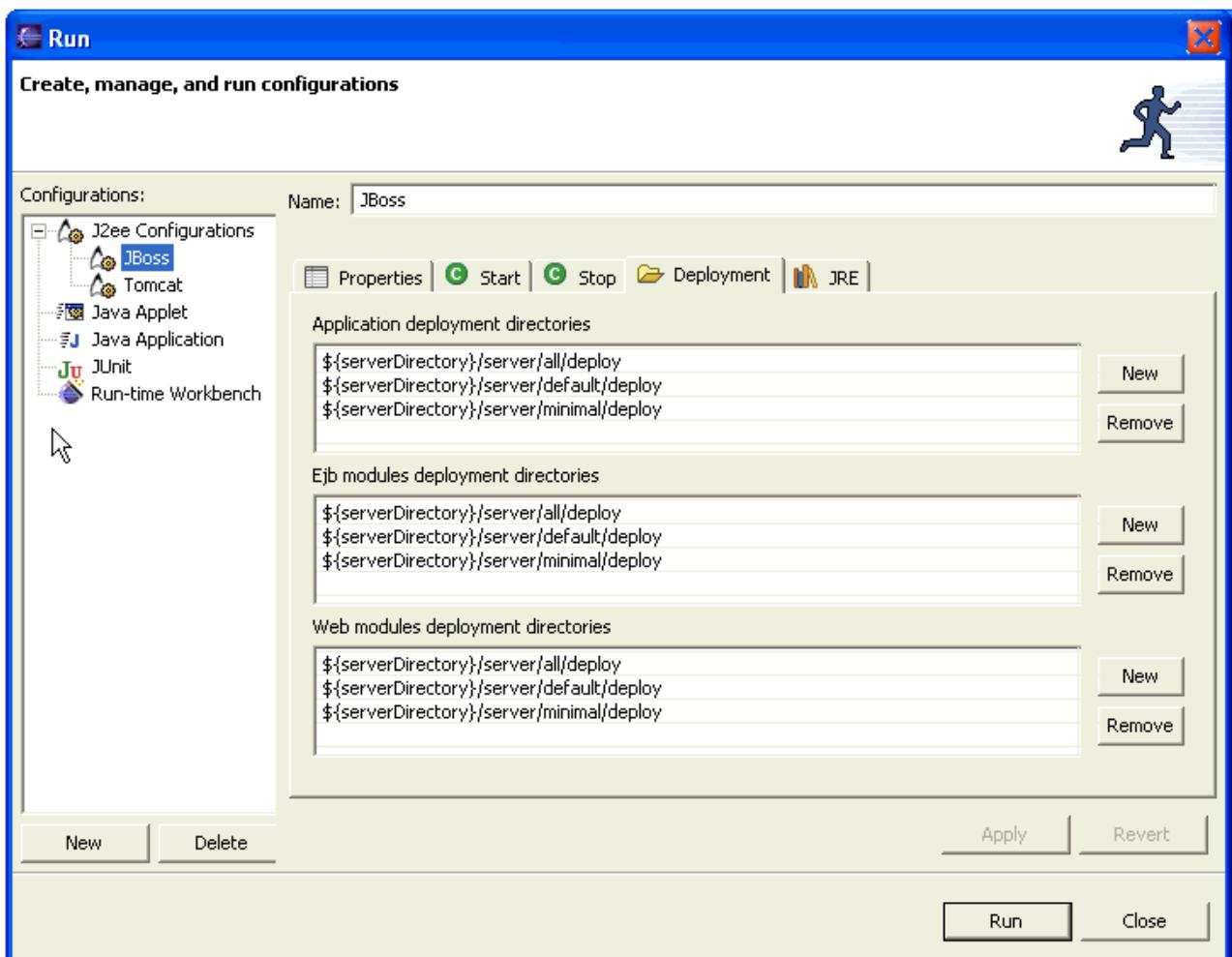
3. Stop tab

You can customize the options to shutdown the server. Default values are available.



4. Deployment tag

You can set the deployment folders. Default values are available.



XDoclet Profile

Concept

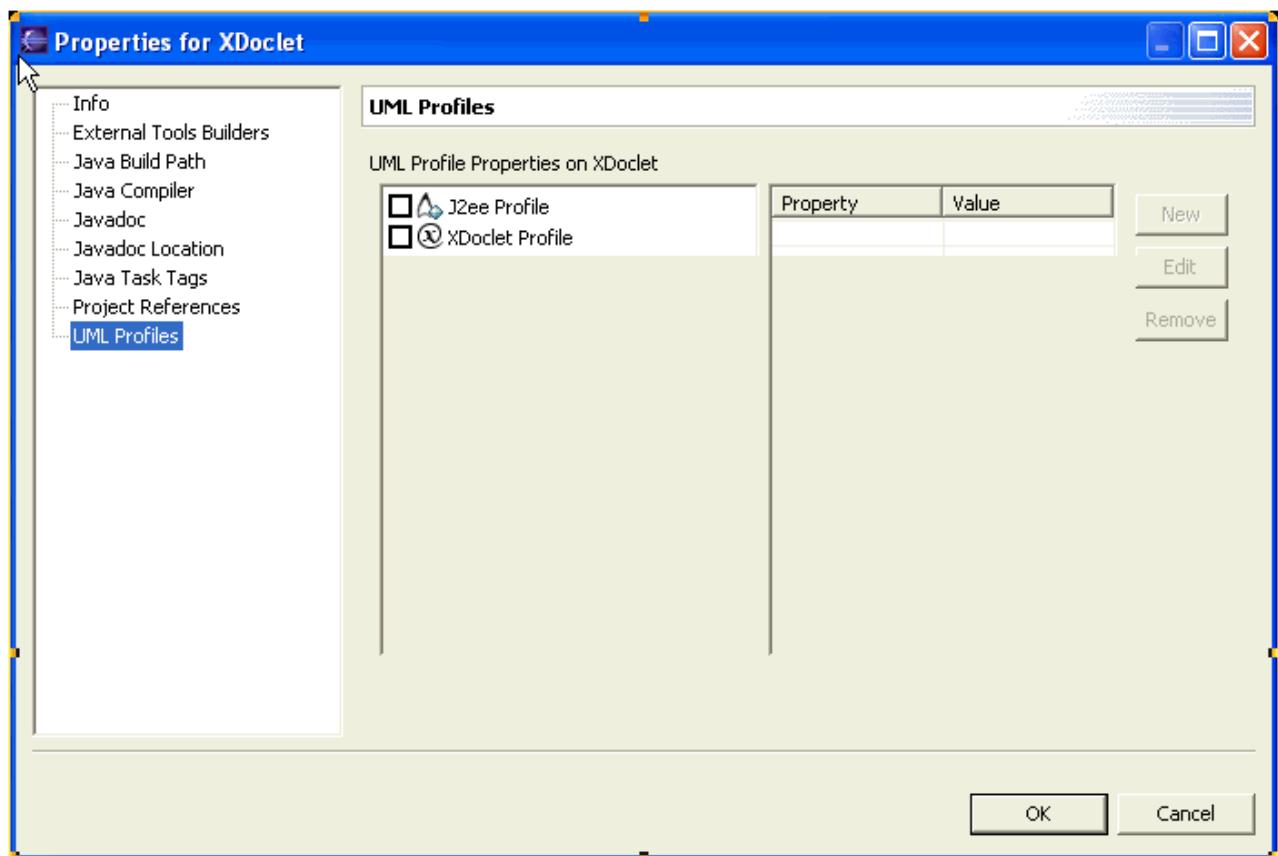
A UML profile expresses the semantics of your application using a set of predefined extensions to UML. UML profiles allow developers to share a common graphical notation and vocabulary, and permit more precise specifications and better documentation on how to use and customize systems. In this XDoclet profile we will present the Class diagram integration.

EclipseUML UML profile for XDoclet allows:

- Full support of any XDoclet functionalities.
- Add your own doclets and customize them.
- Add/Edit doclet tags inside the class diagram editor on:
 - classes
 - interfaces
 - methods
 - attributes

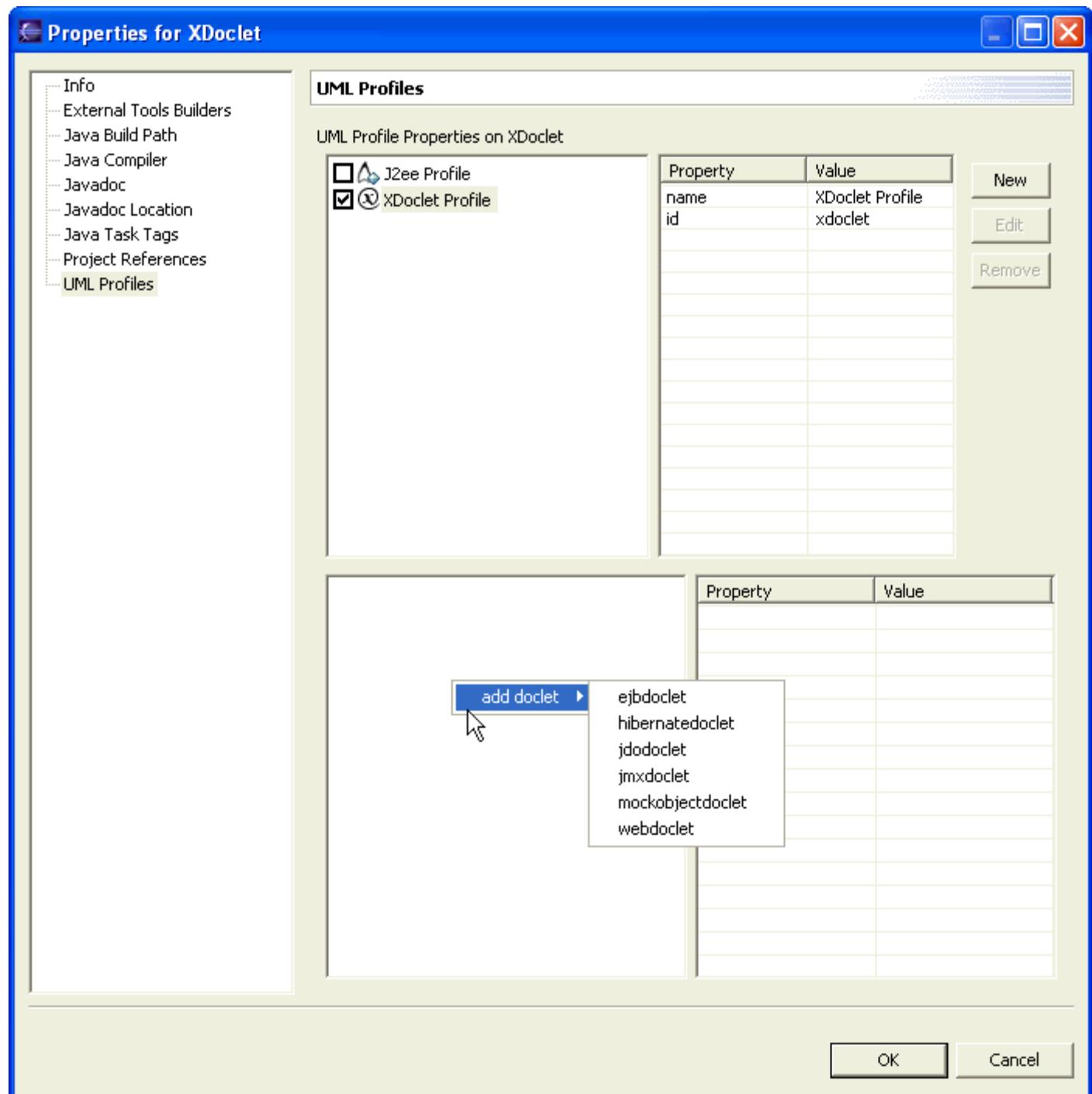
EclipseUML UML profiles work at project level.

In the Package Explorer select the **project > Properties > UML Profiles**

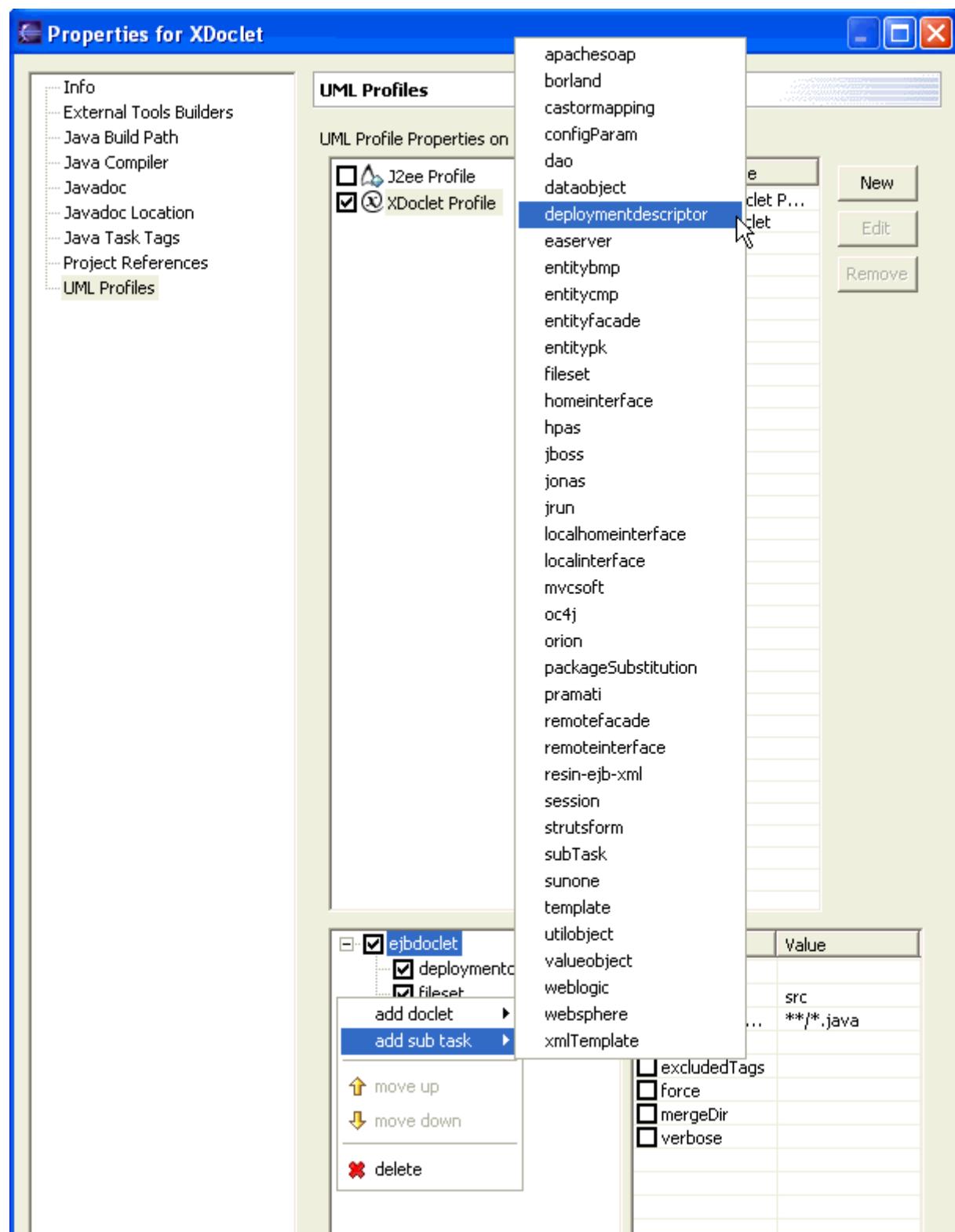


Select XDoclet Profile then add doclet using the pop up menu.

You can add all the standard doclets.

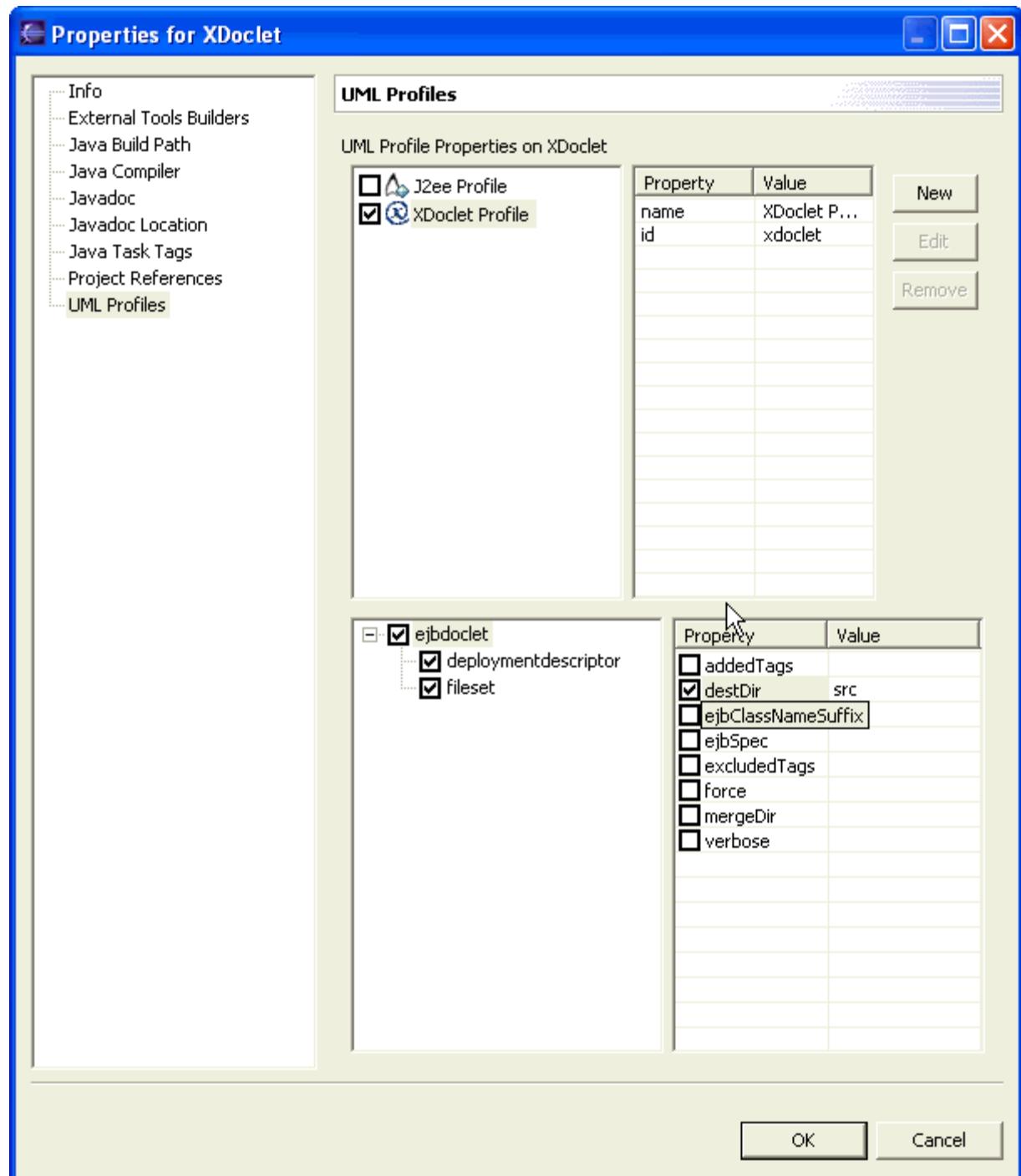


Add some subtasks to the selected doclet.



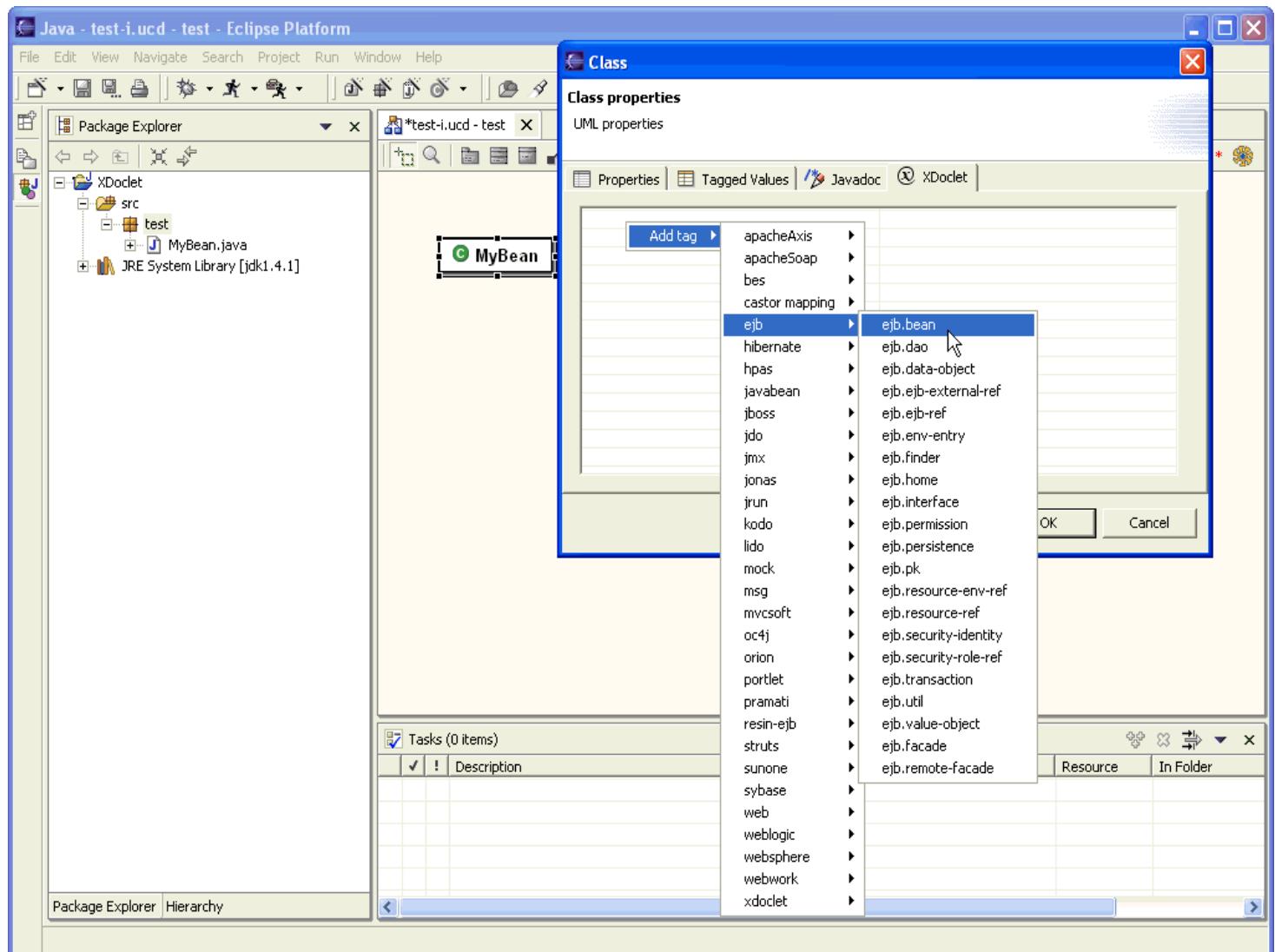
Set up the doclet properties.

For example, the generated code will be saved in the *src* directory because we typed *src* in the value column.

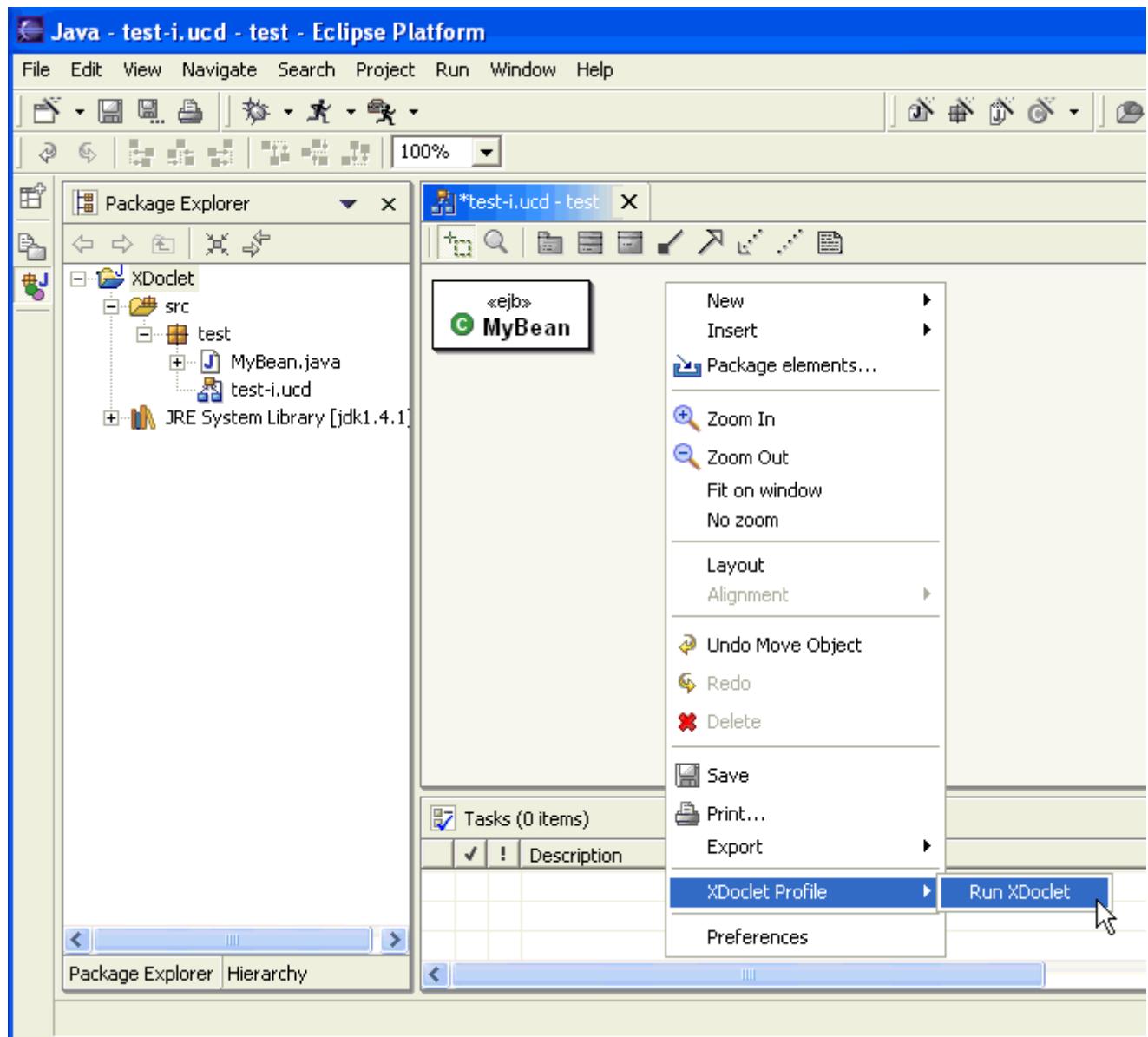


Tags configuration Concept

You need to create a *src* directory > test package > a class. For the purpose of our documentation, we will create a *MyBean* class using the Class Diagram Editor. Select a class in the Class Diagram Editor, open the pop up menu > Properties > XDoclet tab. Use the popup menu to add tags, properties and values.



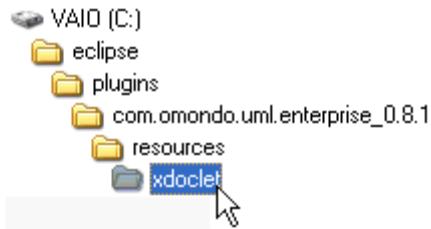
Launch the XDoclet engine by clicking inside the **Class Diagram popup menu > XDoclet Profile > Run XDoclet**.(be sure that no elements are selected in the Class Diagram Editor, because you would open the element pop up menu and not the class diagram pop up menu)
The *Run XDoclet* item will launch the XDoclet engine with the modules choosen in the project's profile configuration.



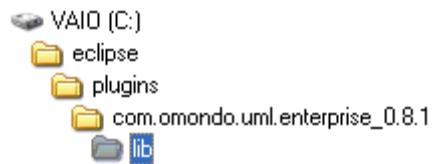
Add your own Doclets

In this section you will learn how to add your own Doclets to the XDoclet profile.

Edit the files in the following directory to add your Doclet:
Eclipse\plugins\com.omondo.uml.std_0.8.1\resources\xdoclet .
The doclets.xml file contains the description of the doclets.
The tags.xml file contains the desription of the XDoclets tags.



Add your jar files in the following directory: eclipse\plugins\com.omondo.uml.std_0.8.1\lib .



Reverse Engineering

Reverse engineering is a processus to rebuild UML model from language implementation. In Java, some information can be retrieved easily by parsing syntaxly Java source code or by decoding class binary file structure, such as:

- Java type class/interface
- Inheritance
- Attributes

Others need a complex code semantic analysis, such as:

- [Getter/setter methods](#)
- [Cardinality of association](#)
- [Element type in a collection](#)
- [Inverse association resolution](#)
- [Qualified association](#)

Precisely, it is necessary to analyze method implementation content to capture them. EclipseUML Personal/Professional/Studio Edition provides a very powerful reverse engineering engine to meet these requirements based on the byte-code analysis. Additionally, it provides following features:

- [Model attribute detection](#)
- Alive dependance detection between classes and packages
- Customizable dependance definition

EclipseUML Personal/Professional/Studio Edition uses the reverse engineering engine in two ways:

- [UML model reverse engineering](#)
- [Dependancy detection](#)

Getter/setter method recognition

UML Reverse engineering detects automatically the getter/setter method of an attribute. It takes into account the prefix and suffix preferences of [JDT](#). The name without prefix and suffix will be used as model name, named as property.

Concrete methods

If methods have the implementations, the UML reverse engineering analyse the code to make sure that the getter method returns this attribute value and the setter changes this attribute value.

For example, with the prefix setting of "f", in the following Java class that contains an attribute name = fCompany, the property name will be "company"

```

public class Employee
{
    private Company fCompany;
    public Company getCompany()
    {
        return fCompany;
    }
    public void setCompany(Company company)
    {
        fCompany = company;
    }
}
public class Company
{
    ...
}

```

After the reverse engineering process, we find UML role name = company, instead of fCompany.



```

public class Employee
{
    /**
     * @uml property=company associationEnd={multiplicity={(0 1)} ordering=ordered * elementType=company.Company}
     */
    private Company fCompany;
    /**
     * @uml property=company
     */
    public Company getCompany()
    {
        return fCompany;
    }

    /**
     * @uml property=company
     */
    public void setCompany(Company company)
    {
        fCompany = company;
    }
}

```

Abstract methods

In case of the abstract method: class abstract method or interface method, there is no code to analyse. So the only solution is to analyse the method name and signature according to JDT definition. The setter method is ignored if a getter method is missing in the class/interface.

Cardinality detection

EclipseUML Reverse Engineering Engine is capable to detect two groups of associations:

- **Cardinality 0..1 or 1**, which correspond to simple reference
- **Cardinality 0...***, which correspond to a container reference

To classify an attribute type to one of the two groups; the key issue is container detection. In Java, there are two categories of containers: `java.util.Collection` and `java.util.Map`. EclipseUML reverse engineering implements a mechanism to recognize not only the interface `java.util.Collection` and `java.util.Map`, but also their subtypes or implementation classes such as `java.util.List`, `java.util.Vector`, `java.util.Hashtable`.

Cardinality 0...1 or 1

For a simple reference, if the attribute is always initialised, the cardinality will be 1. Otherwise, it will be 0..1.

Examples of cardinality 0..1

Before Reverse Engineering	After Reverse Engineering
<pre>public class Employee extends Person { private Company company; public Company() { } }</pre>	<pre>public class Employee extends Person { /** * @uml.property = company * @uml.associationEnd = {multiplicity={(0 1)}} */ private Company company; public Company() { } }</pre>

Example of cardinality 0..1

Before Reverse Engineering	After Reverse Engineering
<pre>public class Employee extends Person { private Company company; public Company(Company company) { this.company = company; } }</pre>	<pre>public class Employee extends Person { /** * @uml.property=company * @uml.associationEnd=multiplicity={(1 1)} */ private Company company; public Company(Company company) { this.company = company; } }</pre>

Cardinality 0...*

A container has always the cardinality 0..*.

Example of collection

Before Reverse Engineering	After Reverse Engineering
<pre>public class Company { private Collection employees = new ArrayList(); ... }</pre>	<pre>public class Company { /** * @uml.property=employees * @uml.associationEnd=multiplicity={(0 -1)} * inverse={company:model.Employee} */ private Collection employees = new ArrayList(); ... }</pre>

Collection element type detection

EclipseUML Reverse engineering engine captures the element type in a collection by analysing content accessing code. For example, the argument of collection method add() may indicate the element type:

```
public class Company

{
    private List employees = new ArrayList();

    public Company(List employees)

    {
        this.employees = employees;
    }

    public void addEmployee(Employee employee)

    {
        employees.add(employee);
    }

    public Collection getEmployees()

    {
        return employees;
    }
}
```

The idea is very simple. But the real use cases are much more complex. The same method may have different implementations.

Using local variable	public void addEmployee(Employee employee) { Collection localVariable = employees; localVariable.add (employee); }
Clall getter	public void addEmployee(Employee employee) { getEmployees().add (employee); }
Using getter and local variable	public void addEmployee(Employee employee)

```

    {
        Collection localVariable = getEmployees();

        LocalVariable.add(employee);

    }
}

```

Of course, these codes can be used outside of the class Company.

According the accessing type of this attribute, we distinguere the analyze mechanism in two categories:

1. Getter category
2. Setter category

The first one allows the forward analyze, the analyzer can follow the execution order. The typical example is the iteration method (see below). It is much easy than **Setter category**, which needs backward analyze mechanism.

1. Getter category

Each accessing method may have a specific way to capture the element type. So we classify all relevant methods as following groups according to the model capture mechanism:

- Add, Remove, Index and Test group
- Get group
- Iteration group

1.1 Add, Remove, Index and Test group

This group includes following methods:

Class	Method
java.util.Collection	boolean add(Object object)
java.util.Vector	void addElement(Object object)
java.util.Vector	void add(int index, Object object)
java.util.Vector	Object set(int index, Object object)
java.util.Vector	Object elementSet (Object object, int index)
java.util.Vector	void setElementAt (Object object, int index)
java.util.Vector	void insertElementAt (Object object, int index)
Class	Method
java.util.Collection	boolean remove(Object object)
java.util.Vector	boolean removeElement(Object object)
java.util.Vector	void removeElement (Object object)
Class	Method
java.util.List	int indexOf (Object object)
java.util.List	int lastIndexOf(Object object)
java.util.Vector	int lastIndexOf (Object object, int index)
java.util.Collection	boolean contains (Object object)

The capture mechanism of this group is simplest one comparing others. We just need identify the method call and capture argument type. For example:

```
public void removeEmployee(Employee employee)
{
    Collection localVariable = employees;
    localVariable.remove(employee);
}
```

or

```
public int getEmployeeIndex(Employee employee)
{
    return employees.indexOf(employee);
}
```

1.2 Get group

This group consists of following methods:

Class	Method
java.util.List	Object get(int index)
java.util.Vector	Object elementAt(int index)
java.util.Vector	Object firstElement()
java.util.Vector	Object lastElement()

This group is a little bit difficult than previous one since we need analyse the next statements to capture the element type in type casting or the operator **instanceof**. For example,

```
public Employee getEmployeeAt(int index)
{
    return (Employee) employees.get(index);
}
```

or

```
public boolean hasEmployeeAt(int index)
{
    Object element = employees.get(index);
    return (element instanceof Employee);
}
```

1.3 Iteration group

This group consists of following methods

Class	Method
java.util.Collection	Iterator iterator ()
java.util.List	Iterator listIterator ()
java.util.List	Iterator listIterator (int index)
java.util.Vector	Enumeration elements ()

This group is more difficult than previous one again since it is necessary to analyse the following cast and **instanceof** statements only after element retrieve call such as `next()` or `nextElement()`. For example,

```
Iterator iterator = company.getEmployees().iterator();

while (iterator.hasNext())
{
    Employee employee = (Employee) iterator.next();
}
```

or

```
for (Iterator iterator = company.getEmployees().iterator(); iterator
hasNext();)

{
    Employee employee = (Employee) iterator.next();
}
```

All casting and operator **instanceof** must be ignored without calling `next()`. Otherwise, the result will be wrong:

```
Iterator iterator = company.getEmployees().iterator();

Address address = (Address) getAddress();

while (iterator.hasNext())
{
    Employee employee = (Employee) iterator.next();
}
```

2. Setter category

When the code assigns a filled container to this attribute, for example,

```
List employees = new ArrayList();  
employees.add(new Employee());  
Company company = new Company (employees);
```

or

```
List employees = new ArrayList();  
employees.add(new Employee());  
company.setEmployees(employees);
```

it is necessary to perform the semantic analyze following inverse execution order. EclipseUML Personal/Professional/Studio edition implements this sophistic mechanism. It is used only when the first category calls are missing.

Inverse association resolution

EclipseUML Reverse engineering uses following rules to resolve inverse associations:

1. If Class A has an association “b” of type B and Class B hasn’t any association of type A, there isn’t inverse association.
2. If Class A has an association “b” of type B and Class B has only one inverse association “a” of type A, we set up the two associations as inverse associations.
3. If Class A has an association “b” of type B and Class B has more than one association of type A (for example, “a1” and “a2”), we try to resolve this conflict by analysing the accessing of the two couple of attributes: “b” and “a1”, “b” and “a2”.
 - a. If there is only one method that accesses one couple of attributes directly or via getter/setter, this couple of attributes will be considered as inverse associations.
 - b. If there is more than one method that accesses both couples of attributes, we use collected statistic to resolve this conflict.

For example:

```
public class Project

{

private HashMap teams = new HashMap ();

public void addTeamMember(String name, Employee member) {

    Collection team = (Collection) teams.get(name);

    if (team == null) {

        team = new ArrayList();

        teams.put(name, team);

    }

    team.add(member);

    member.setProject(this);

}

}
```

The attribute accesses are in bold. So the associations implemented by the attribute `teams` and `project` are inverse associations.

Qualified association detection

EclipseUML Reverse engineering engine can recognize the `java.util.Map` and its implementation classes such as `java.util.HashMap` and `java.util.Hashtable`. It can capture not only the key and value type, but also the element type in the associated value if it is a collection.

In the following illustration, we use the Company and Project class as example:

```
public class Company

{
    private Hashtable projects = new Hashtable ();

    public Company(Hashtable projects)

    {
        this.projects = projects;
    }

    public Hashtable getProjects()

    {
        return projects;
    }

    public void setProjects (Hashtable projects)

    {
        this.projects = projects;
    }
}
```

According the accessing type of this attribute, we distinguee the analyze mechanism in two categories:

1. [Getter category](#)
2. [Setter category](#)

The first one allows the forward analyze, the analyzer can follow the execution order. The typical example is the iteration method (see below). It is much easy than [Setter category](#), which needs backward analyze mechanism.

1. Getter category

Each accessing method may have a specific way to capture the element type. So we classify all relevant methods as following groups according to the model capture mechanism:

- Put, Remove, Index and Test group
- Get group
- Iteration group

1.1 Put, Remove and Test group

This group includes following methods:

Class	Method	Recognition
java.util.Map	Object put(Object key, Object value)	Key and value
Class	Method	Recognition
java.util.Map	Object remove(Object object)	Key
Class	Method	Recognition
java.util.Map	boolean containsKey (Object object)	Key
java.util.Map	boolean containsValue (Object object)	Value
java.util.Hashtable	boolean contains (Object object)	Value

The capture mechanism of this group is simplest one comparing others. We just need identify the method call and capture argument type. For example:

```
public void putProject(String name, Project project)
{
    projects.put(name, project);
}
```

1.2 Get group

This group consists of following methods:

Class	Method	Recognition
java.util.Map	Object get(Object object)	Key and value

This group is a little bit difficult than previous one since we need analyse the next statements to capture the element type in type casting or the operator `instanceof`. For example,

```
public Project getProjectAt(String name)
{
    return (Project) projectsMap.get(name);
}
```

1.3 Iteration group

This group consists of following methods

Class	Method	Recognition
java.util.Map	Iterator values ()	Value
java.util.Map	java.util.Set keySet() followed by iterator()	Key
java.util.Map	Collection values() followed by iterator()	Value
java.util.Hashtable	Enumeration keys ()	Key
java.util.Hashtable	Enumeration elements ()	Value

This group is more difficult than previous one again since it is necessary to analyse the following cast and **instanceof** statements only after element retrieve call such as `next()` or `nextElement()`. For example,

```
Iterator iterator = company.getProjects().values().iterator();

while (iterator.hasNext())

{
    Project project = (Project) iterator.next();
}
```

or

```
for (Iterator iterator = company.getProjects().keySet().iterator(); iterator
hasNext();)

{
    String name = (String) iterator.next();
}
```

All casting and operator **instanceof** must be ignored without calling `next()`. Otherwise, the result will be wrong:

```
Iterator iterator = company.getProjects().values().iterator();

Address address = (Address) getAddress();

while (iterator.hasNext())

{
    Project project = (Project) iterator.next();
}
```

If the value is a collection, the element type mechanism will be used for deep analyze. Here is an example of the association teams between Employee and Project:

```

public class Project

{
    private HashMap teams = new HashMap ();

    public void addTeamMember(String name, Employee member) {

        Collection team = (Collection) teams.get(name);

        if (team == null) {

            team = new ArrayList();

            teams.put(name, team);

        }

        team.add(member);

        member.setProject(this);

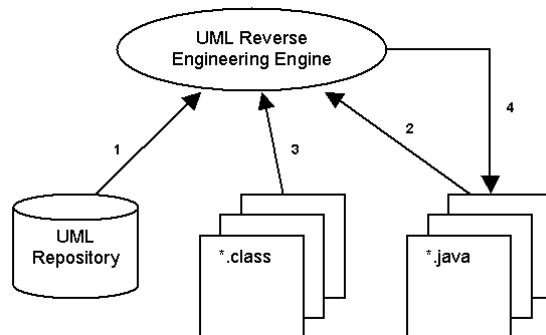
    }
}

```

UML Model Reverse Engineering

UML model reverse engineering is a complex processus. It provides not only a byte-code reverse engineering engine, but also an migration capability from Free edition to Personal/Professional/Studio editions. In terms of data flow, it read from :

- UML Repository created by EclipseUML free edition: it parts of the migration processus
- Existing UML Model stored as annotation in Java source
- the UML Model produced by bytecode engine



And finally, the results are stored back as annotation into Java sources.

The Reverse engineering processus consists of three steps:

1. Setup preferences
2. Launch Reverse Engineering
3. Select model sources

1. Setup Preferences

First, it is very important to set up some Preferences in Eclipse:

- Eclipse platform perferences
- JDT preferences

These preferences influence the Reverse engineering results.

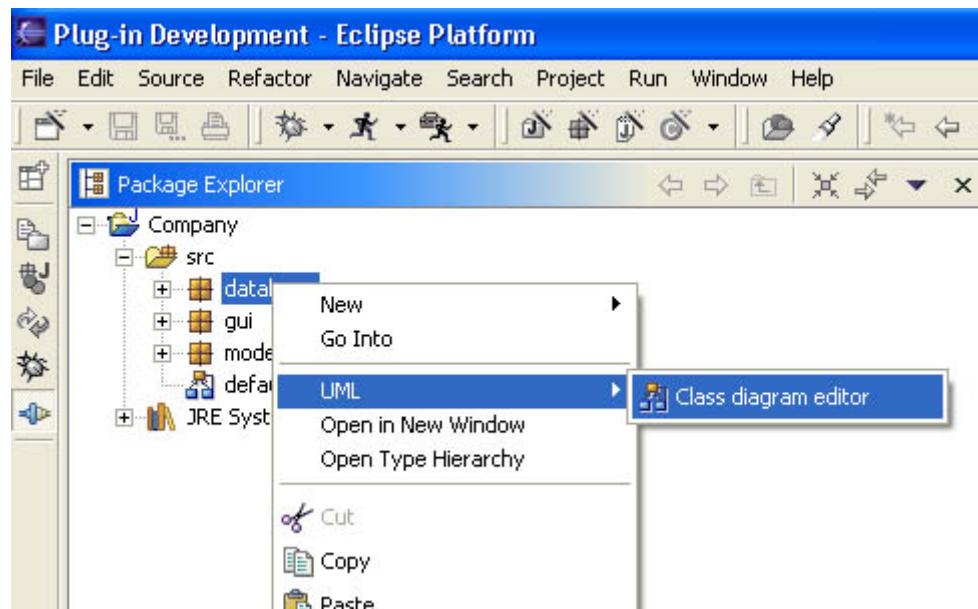
2. Launch Reverse Engineering

There are two ways to invoke UML Model reverse engineering:

1. Automatical Reverse Engineering
2. Explicit Reverse Engineering

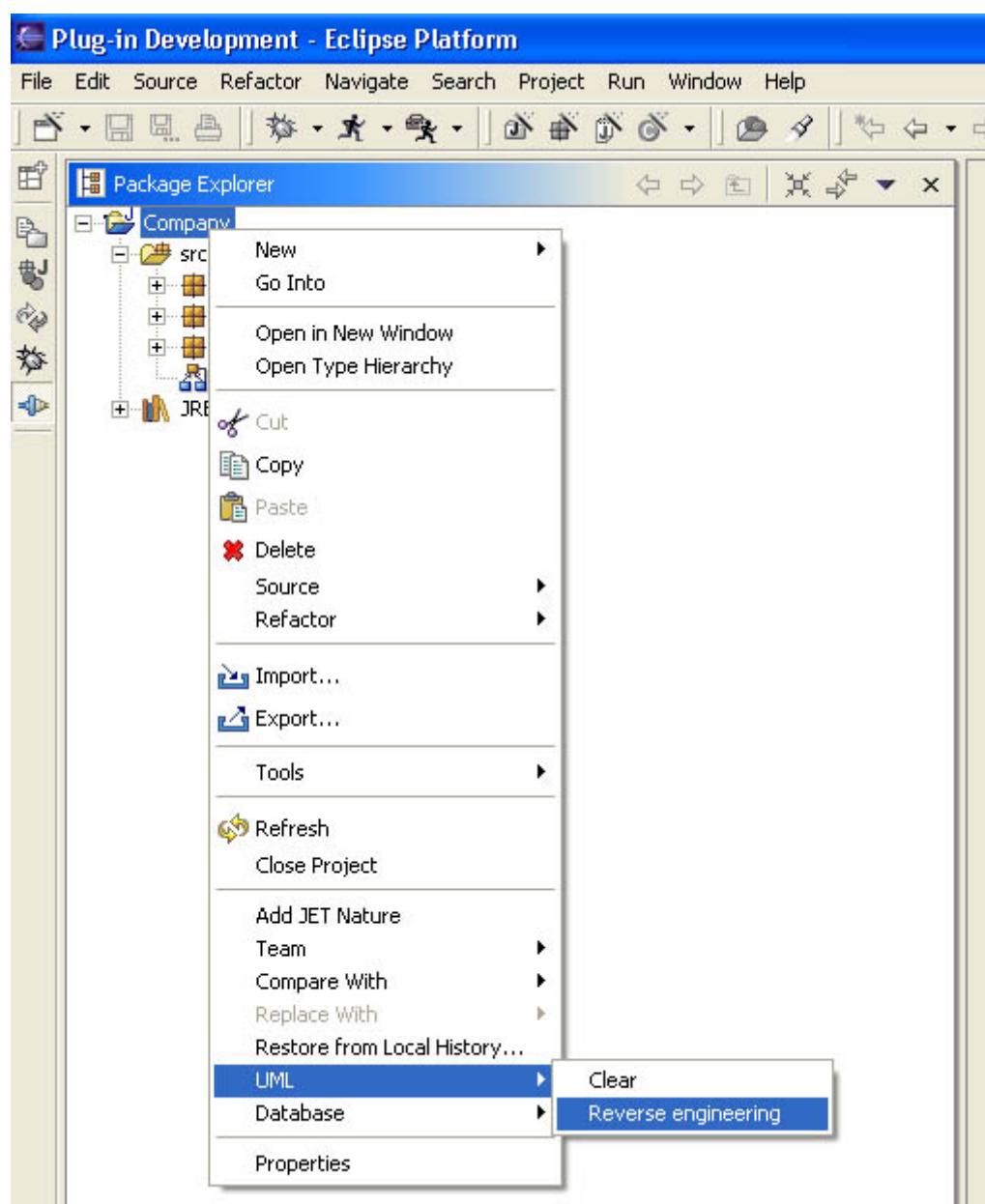
2.1 Automatical Reverse Engineering

When you create the first diagram in a Java project, the Reverse Engineering is started automatically. There are two possibilities to create a class diagram: using [UML new wizard](#) or calling short cut menu. For the second solution, you need select one package in the Package explorer and right click on the mouse to call the context menu **UML>Class diagram editor** as following:



2.2 Explicit Reverse Engineering

It is also possible to perform the Reverse Engineering whenever you want by selecting the Java Project in Package explorer and then right click to call the context menu: **UML>Reverse engineering:**



3. Model Source Options

The Reverse Engineering has different options if the Java project contains already UML models from EclipseUML free edition.

3.1 Reverse engineering without migration

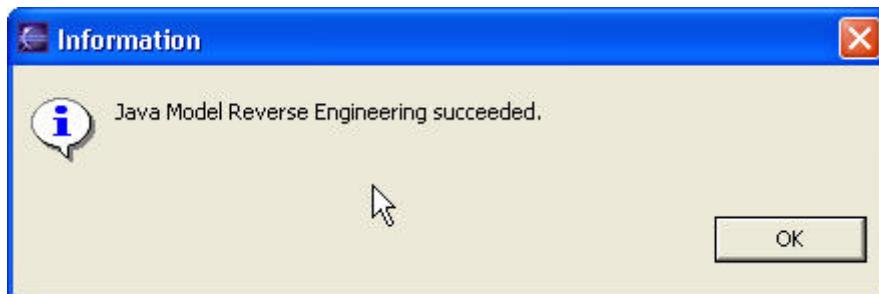
If the Java project has never touched by EclipseUML free edition, when you Reverse engineering is activated, you will get following option dialog.

If you already have existing UML annotations in your project, you will be able to ignore them by selecting the "*Ignore existing UML annotation*" checkbox.

This checkbox will not appear if the project doesn't have UML annotation.,

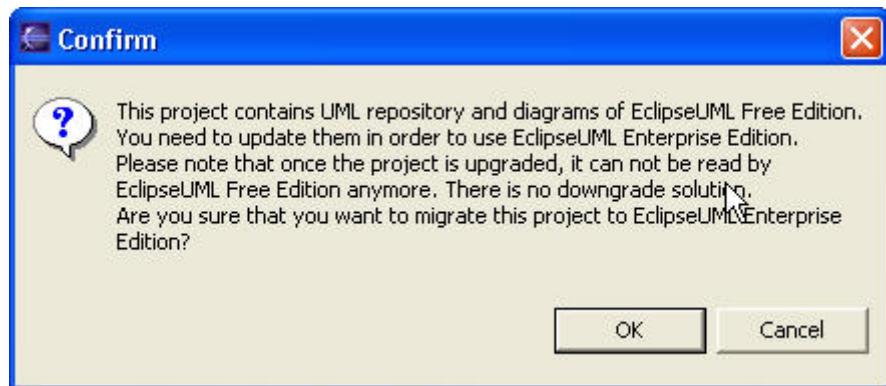


It is possible to change some options to select the model sources.



3.2 Reverse with migration

If the project contains the UML repository created within EclipseUML free edition, a warning dialog will alarm that the Reverse engineering is irreversible.

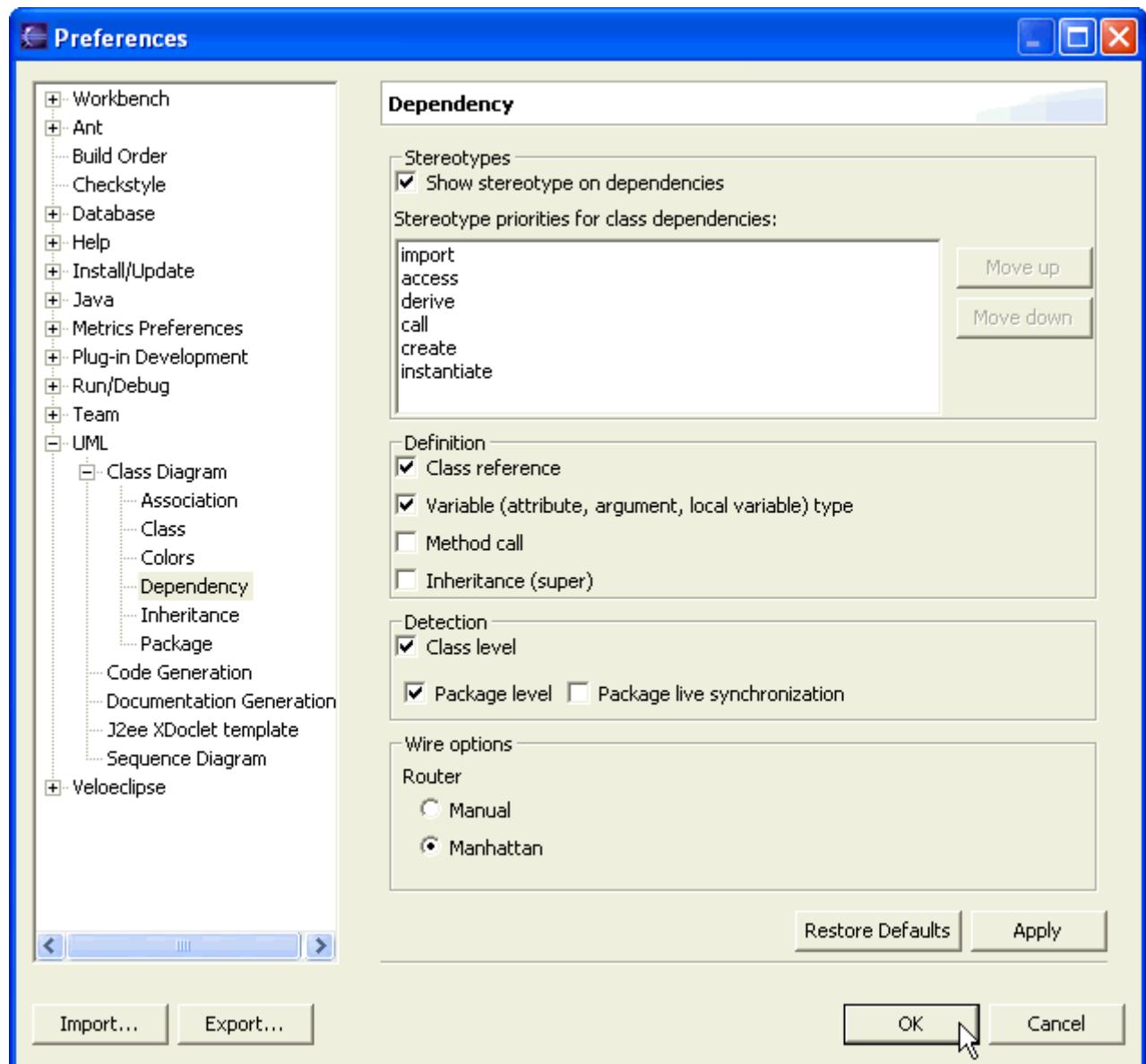


If you confirm the operation, you will get following option dialog:



Dependency

Dependencies can be customized from the preferences (UML > Class Diagram > Dependency).



Documentation Generation

Launching the documentation generation

The documentation generation can be launched from the pop-up menu or from the menu bar after selecting a java element in the package explorer. A wizard dialog appears.

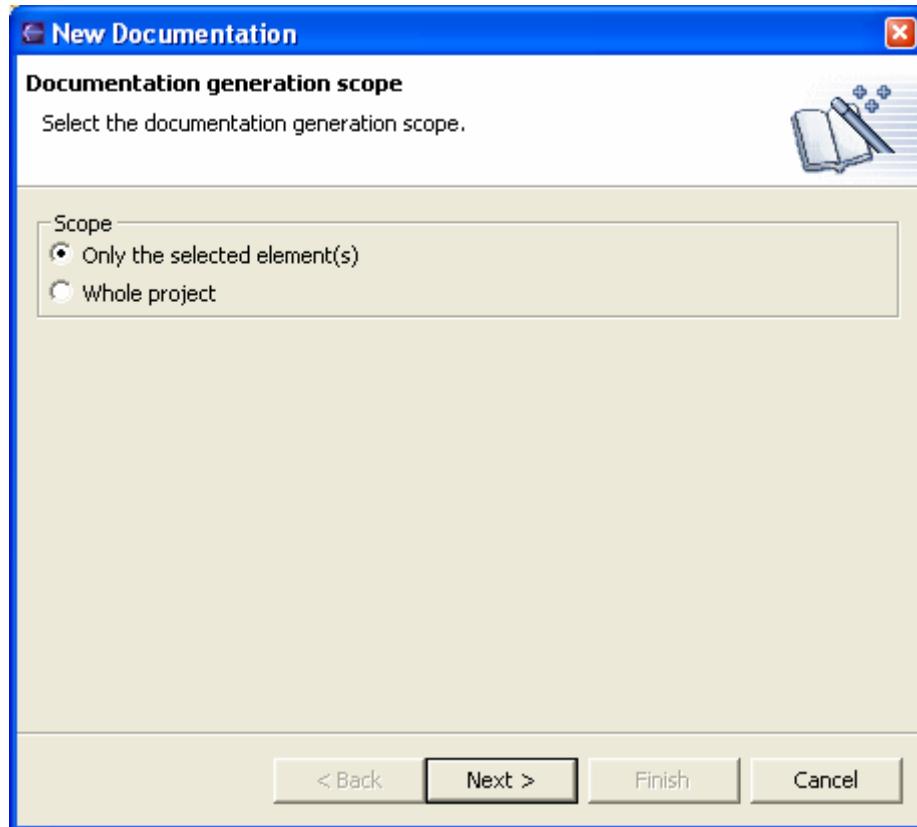
Selection

The documentation generation can be launched from the pop-up menu or from the menu bar after selecting a java element in the package explorer. The generated documentation will be dedicated either to this java element or to the whole java project. A wizard dialog appears.

Documentation Generation Wizard

Selection

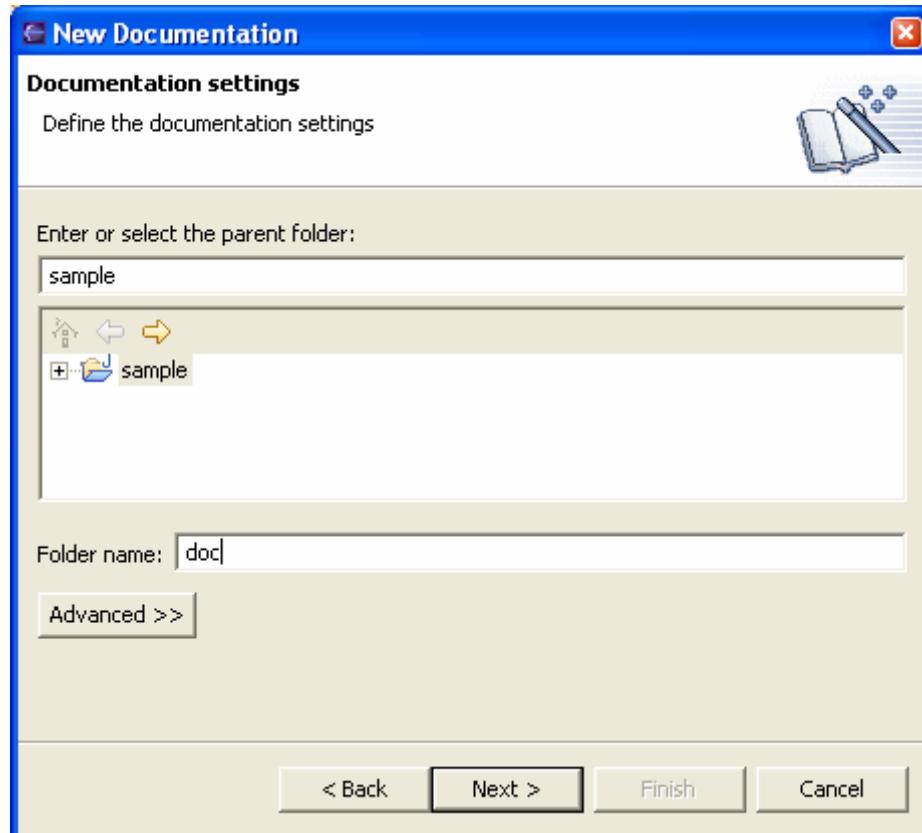
On the first step of the wizard, you can choose either to work on your selection or to work on the whole project.



Documentation directory

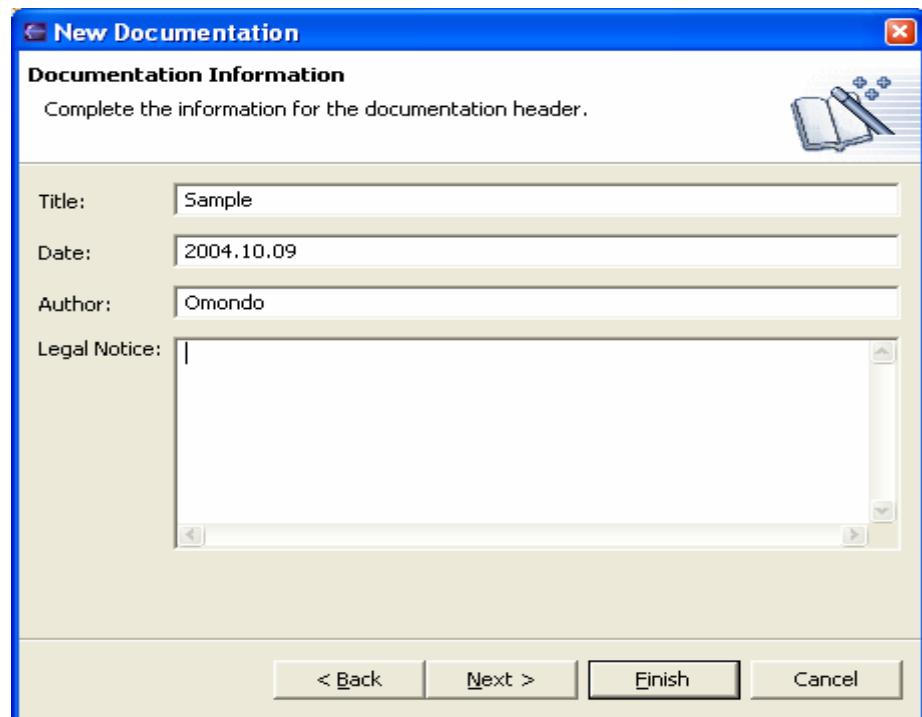
On the second step of the wizard, you must select the directory where the documentation will be generated.

It can be either an existing folder or a new one or a linked one.



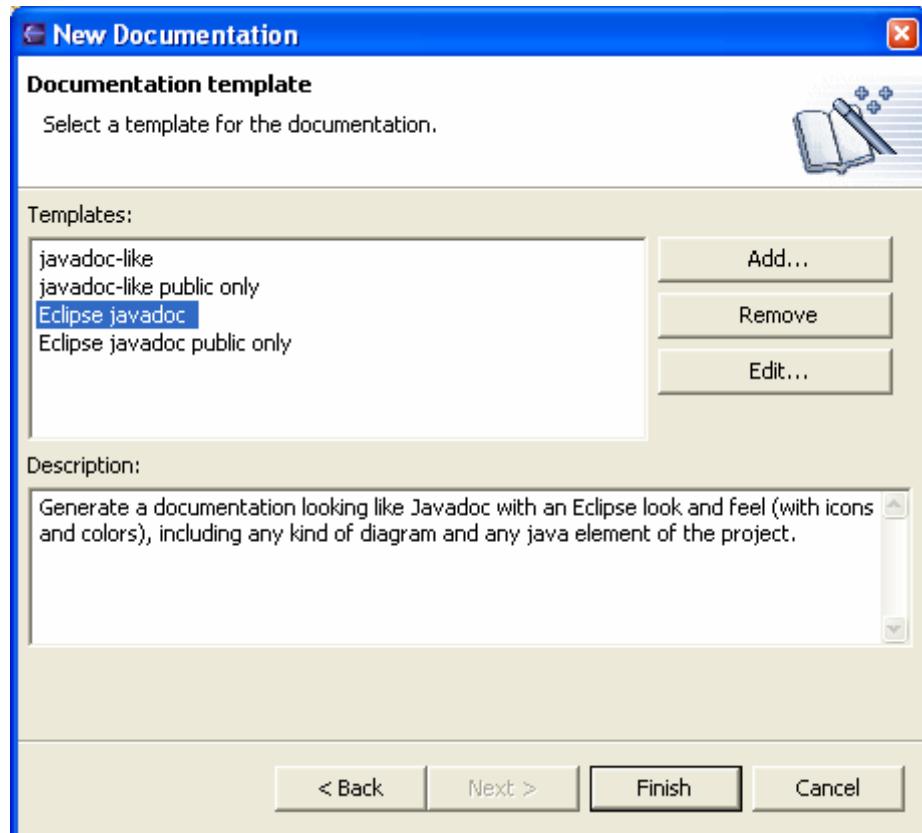
Documentation information

On the third step of the wizard, you can edit the title and date of the documentation. Author and authoring (legal notice, licence, ...) information can be added



Templates

On the last step of the wizard, you must select a [template](#) for the documentation generation. Each template define which kind of element will be inserted in the generated documentation and how it will be formated. So if you selected a single class, you won't have any template proposed by default and will have to define one by yourself.



Generation

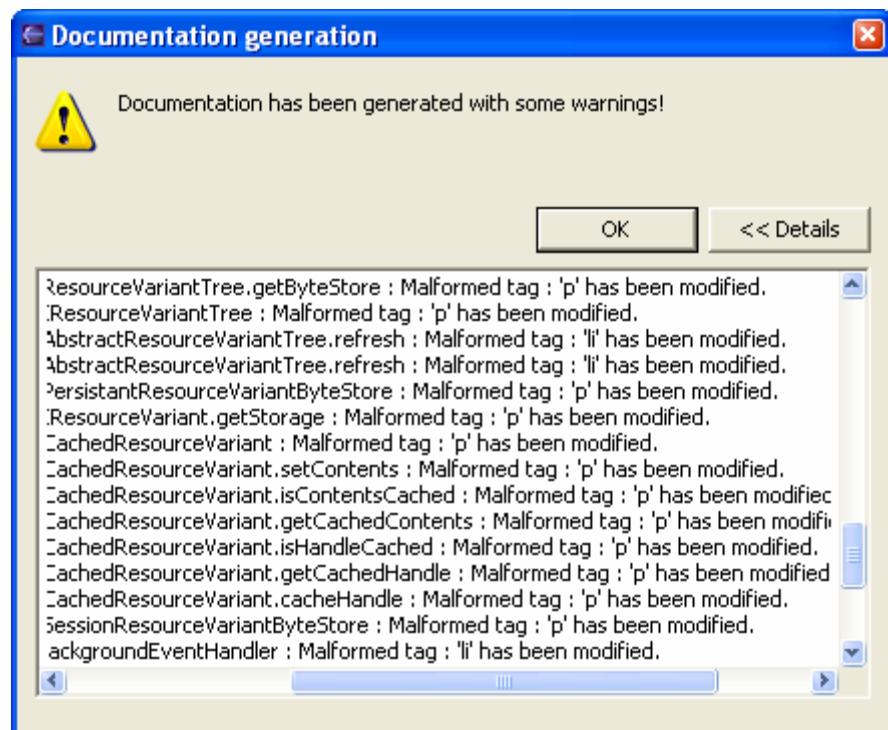
The documentation is generated through several steps :

- Documentation elements analysis : the elements to include in the documentation are listed
- Elements javadoc analysis : the javadocs are analysed, special characters are changed to numeric entities and errors are fixed.
- Diagram images generation : the UML diagram to include in the documentation are open and images are created.
- Documentation format : the final XSL transformation is processed to create the final documentation.

Javadoc errors management

If any error is found in the javadoc, it is fixed in the generated documentation. For example, malformed tag-embedments are skipped and unclosed tags are closed. < and > characters which are not tag delimiters are changed to numeric entities.

The processed modifications are listed at the end of the generation. Original source files are not modified.



Result

Here is a sample of a generated documentation :

The screenshot shows a Microsoft Internet Explorer window displaying a JavaDoc index page. The title bar reads "D:\projects\workspace.RC1\org.eclipse.team.core\doc\index.html - Microsoft Internet Explorer". The menu bar includes "Fichier", "Edition", "Affichage", "Favoris", "Outils", and "?". The toolbar contains icons for Back, Forward, Stop, Refresh, Home, Search, Favorites, Media, and Print. The address bar shows the URL "D:\projects\workspace.RC1\org.eclipse.team.core\doc\index.html". Below the toolbar is a navigation bar with tabs: "Overview", "Package", "Class", and "Diagram". The "Overview" tab is selected.

org.eclipse.team.core

[All](#)

Packages

- [org.eclipse.team.core](#)
- [org.eclipse.team.core.subscribers](#)
- [org.eclipse.team.core.synchronize](#)
- [org.eclipse.team.core.variants](#)
- [org.eclipse.team.internal.core](#)
- [org.eclipse.team.internal.core.simpleAccess](#)
- [org.eclipse.team.internal.core.streams](#)

All Diagrams

- [core-ai](#)
- [subscribers-ai](#)
- [synchronize-i](#)
- [variants-ai](#)
- [core-ai](#)
- [simpleAccess-ai](#)
- [streams-ai](#)
- [subscribers-ai](#)

All Classes

- [AbstractResourceVariantTree](#)
- [AndSyncInfoFilter](#)
- [Assert](#)
- [AssertionFailedException](#)
- [AutomergableFilter](#)
- [BackgroundEventHandler](#)
- [BaseResourceVariantByteStore](#)
- [BatchingLock](#)
- [CachedResourceVariant](#)
- [CloudResourceVariant](#)

org.eclipse.team.core Packages

org.eclipse.team.core	
org.eclipse.team.core.subscribers	
org.eclipse.team.core.synchronize	
org.eclipse.team.core.variants	
org.eclipse.team.internal.core	
org.eclipse.team.internal.core.simpleAccess	
org.eclipse.team.internal.core.streams	
org.eclipse.team.internal.core.subscribers	

org.eclipse.team.core 2004.10.09

[Overview](#) [Package](#) [Class](#) [Diagram](#)

[Poste de travail](#)

Templates

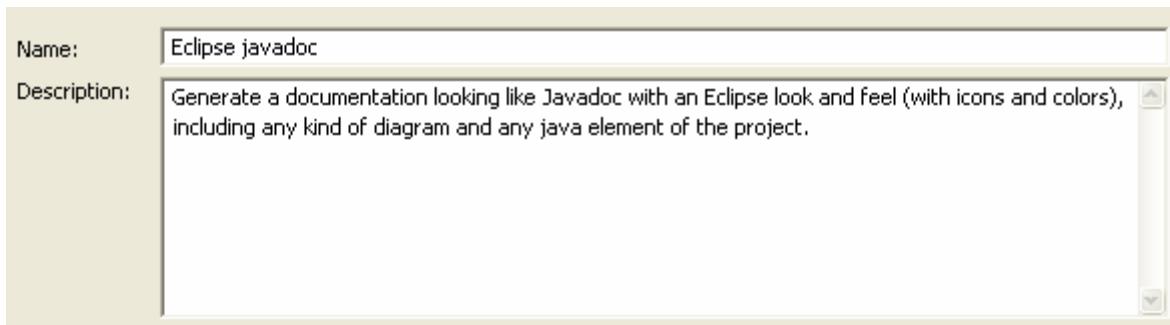
The documentation generation templates are containing three kinds of information :

- [Information on the template](#)
- [Content of the generated documentation](#)
- [Format of the generated documentation](#)

These templates can be edited either from the preferences or from the documentation generation wizard.

Information

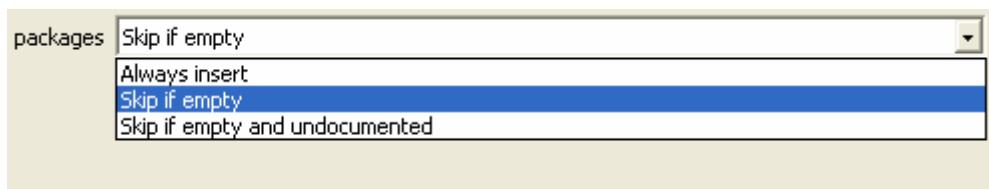
A template have a name and a description to explain what is it dedicated for.



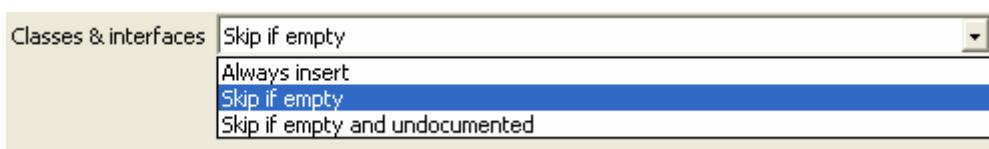
Content

When editing a template, it is possible to define what should be included in the generated documentation.

- Packages



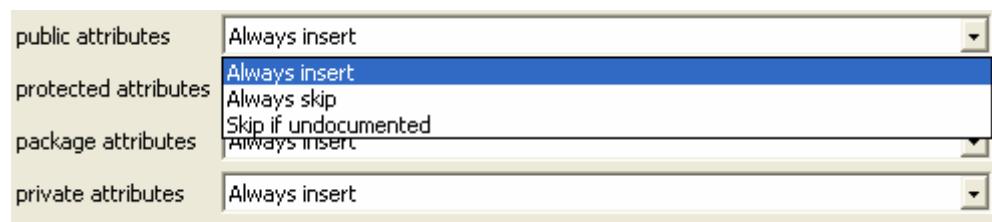
- Class and Interfaces



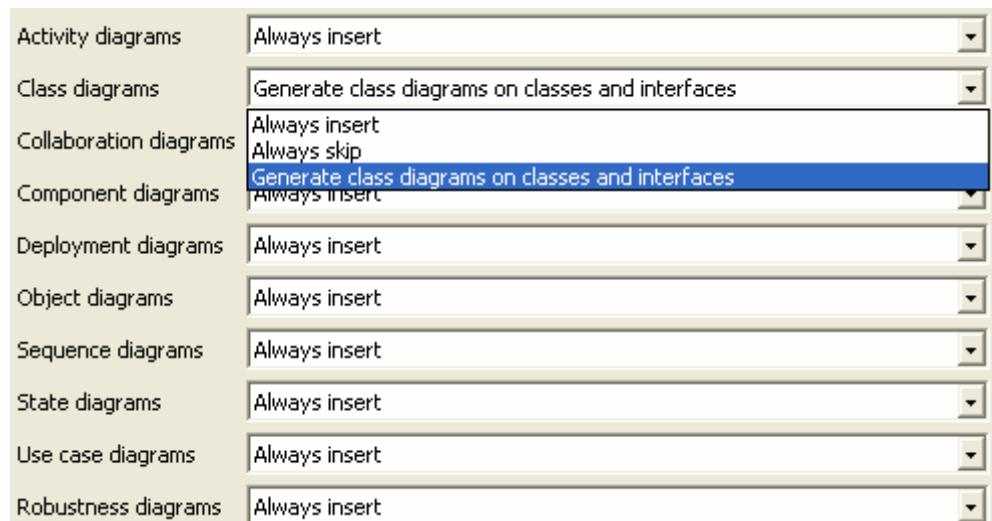
- Methods



- Attributes



- Diagrams



Class diagram dedicated to a class can be automatically generated.

Format

The final format of the generated documentation depends on a XSL transformation. The XSLT template to use must be indicated.



For now, the DTD used for the XSL tranformation is not published. It will be enhanced and published on a furthercoming version.

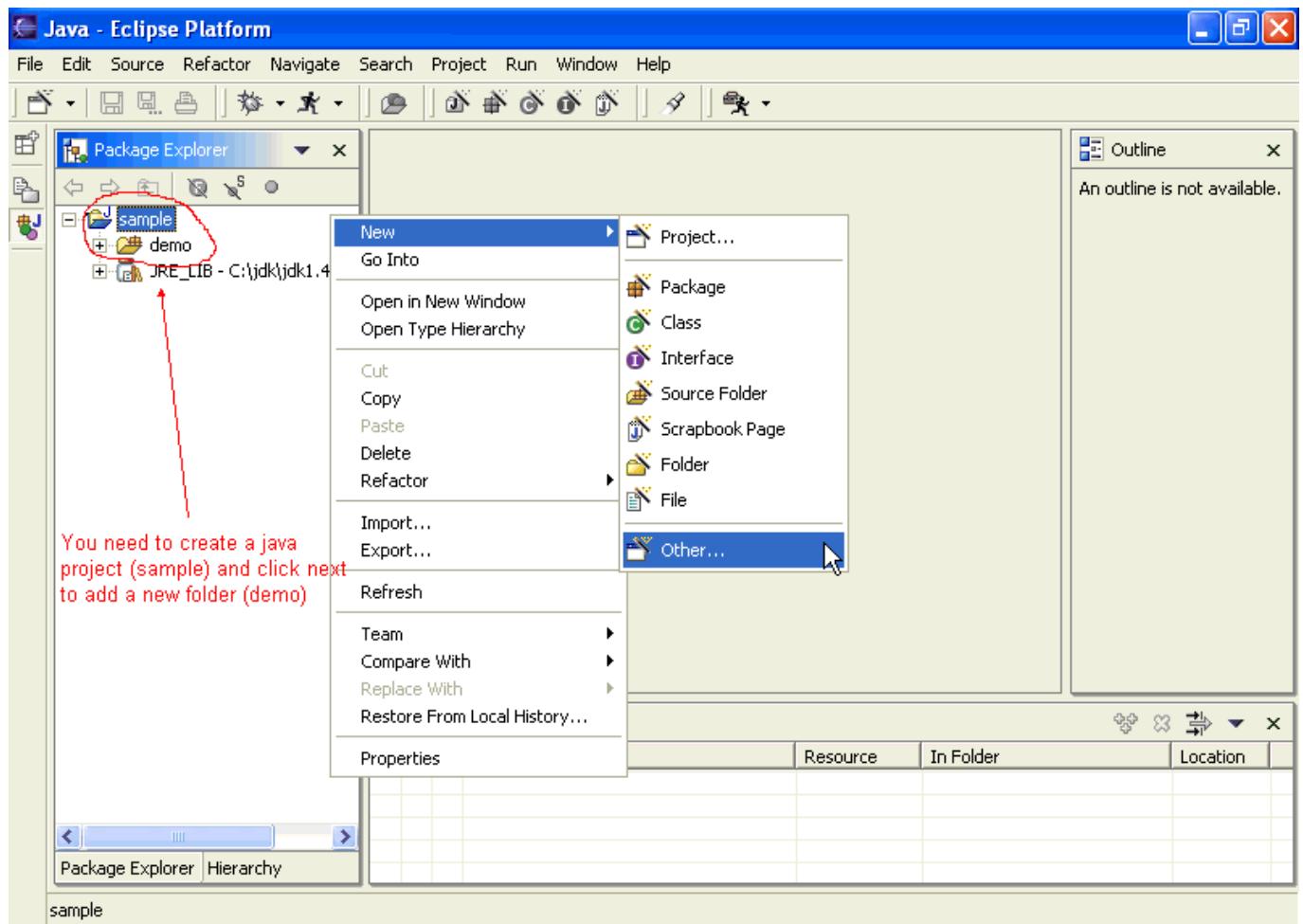
Two default XSLT templates are provided :

- Javadoc-like documentation
- Eclipse Javadoc-like documentation (with Eclipse icons and decoration added to a standard-style javadoc)

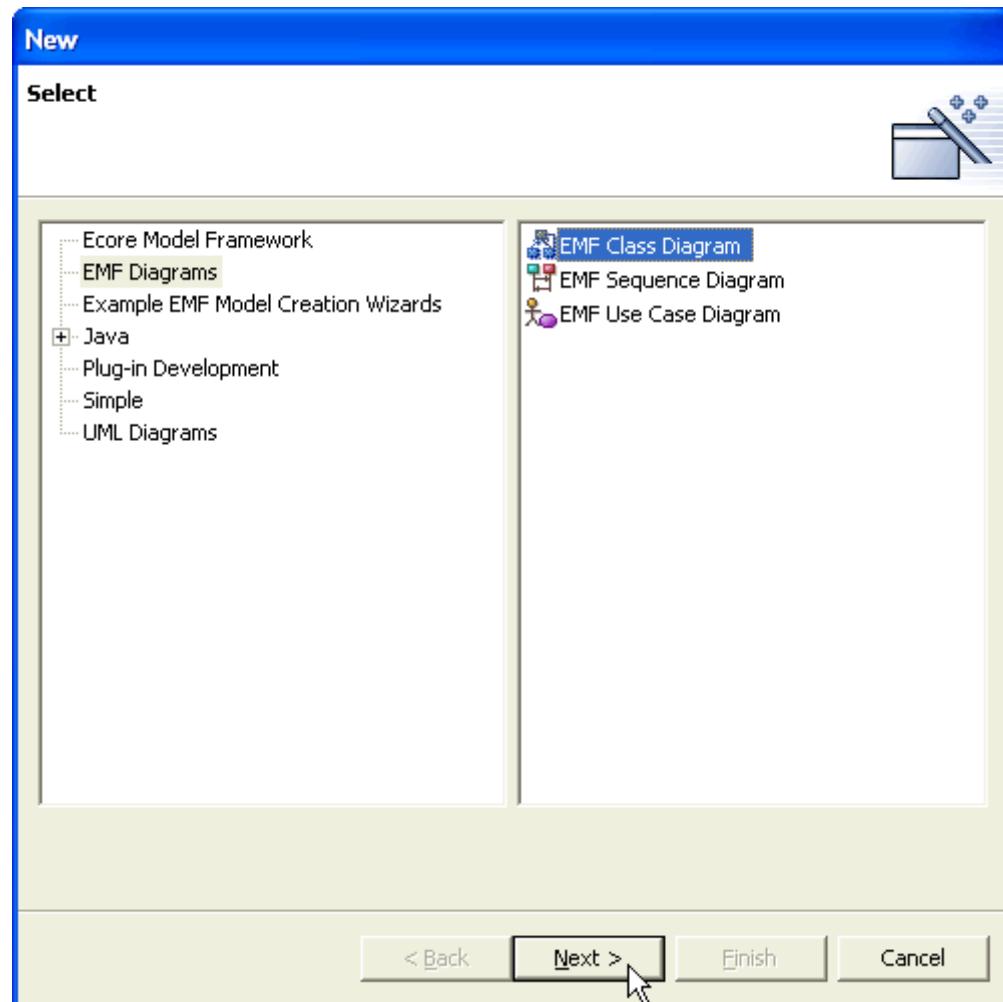
Getting Started with EMF

Class Diagram Creation

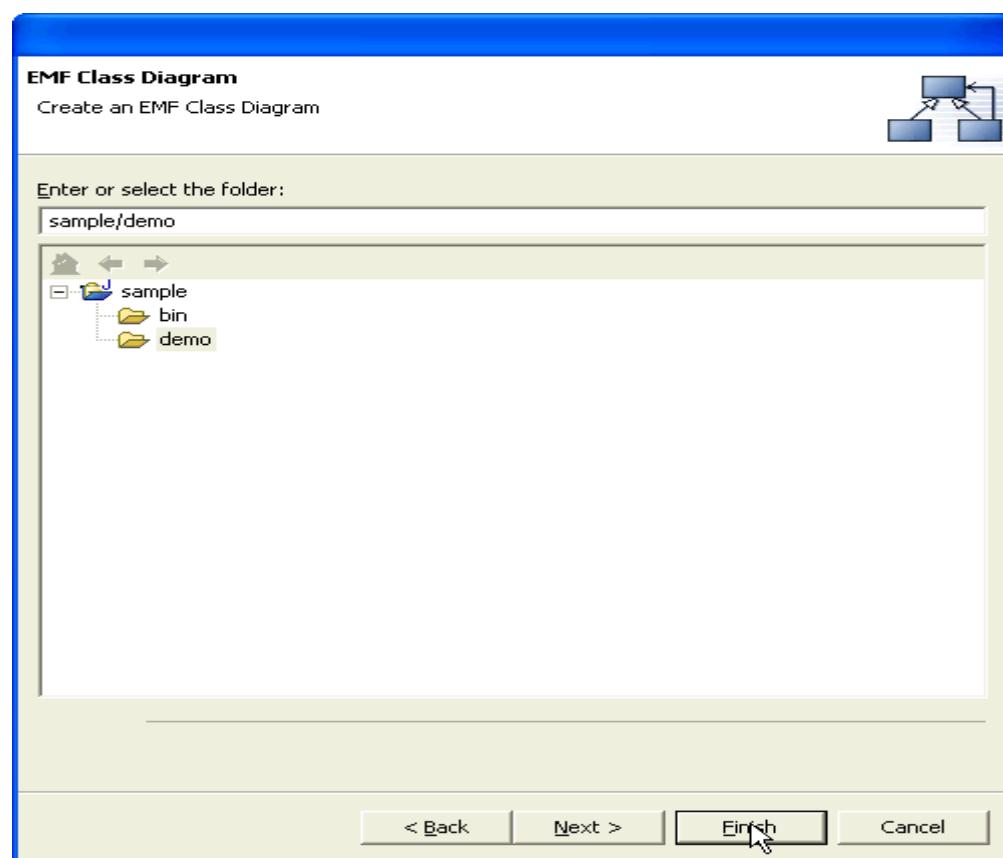
To create a new class diagram, select *New > Others* in the contextual menu.



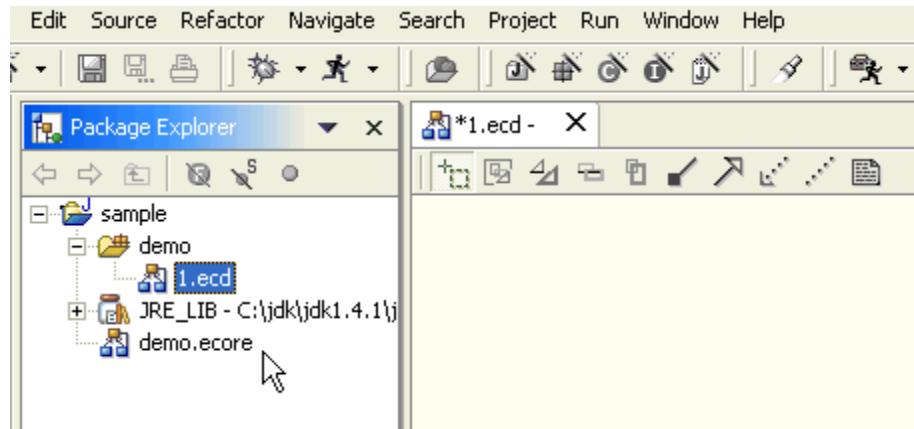
Then select *UML > EMF class diagram*.



Select the package you want to work in.

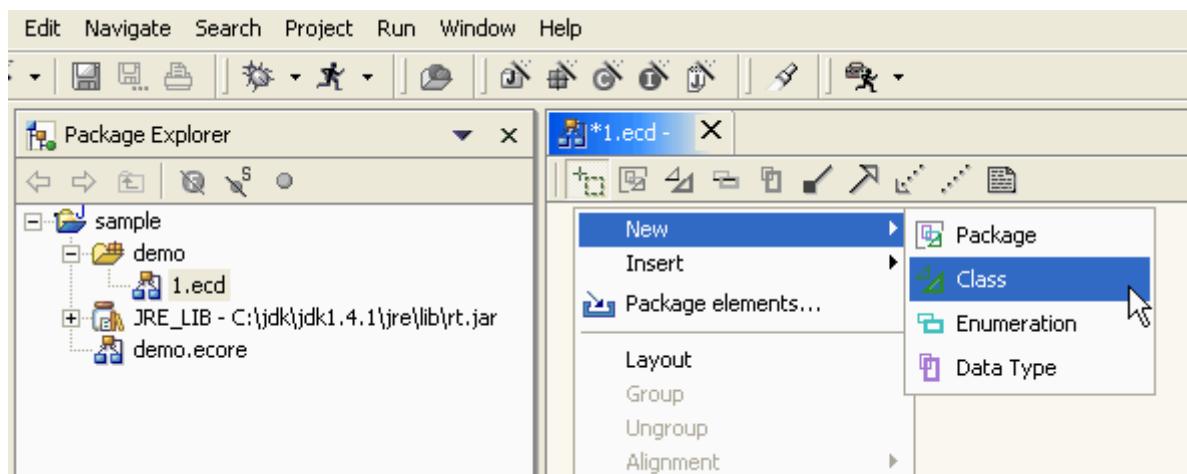


Your new EMF class diagram area has been created.

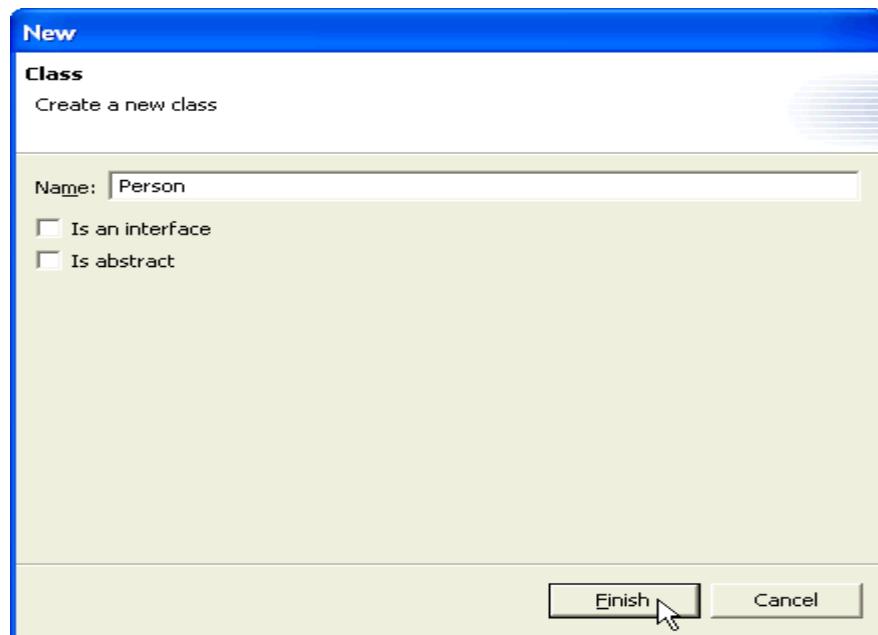


Class and Interfaces

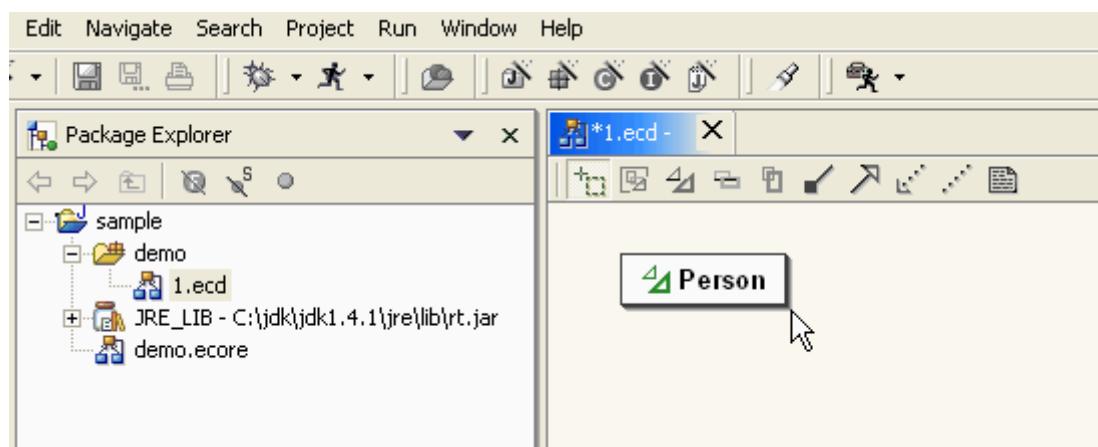
To create a new class or a new interface directly in your EMF class diagram editor, select *New > Class* from the EMF class diagram editor contextual menu.



The EclipseUML class creation dialog appears.

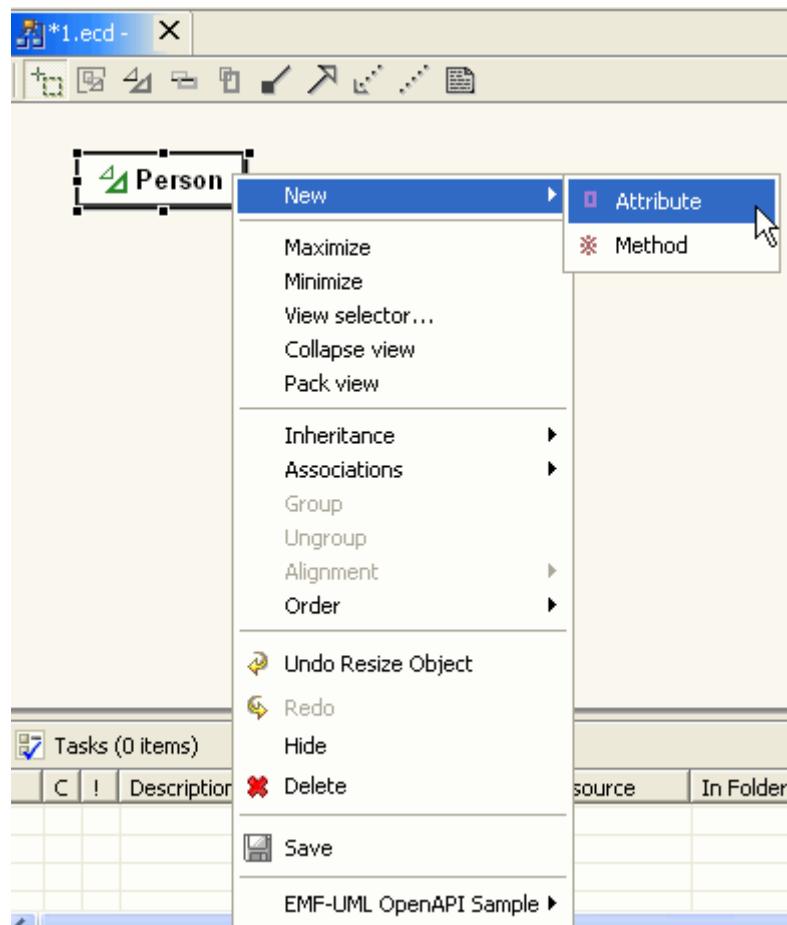


After validating the dialog box, the class appears immediately in the class diagram.

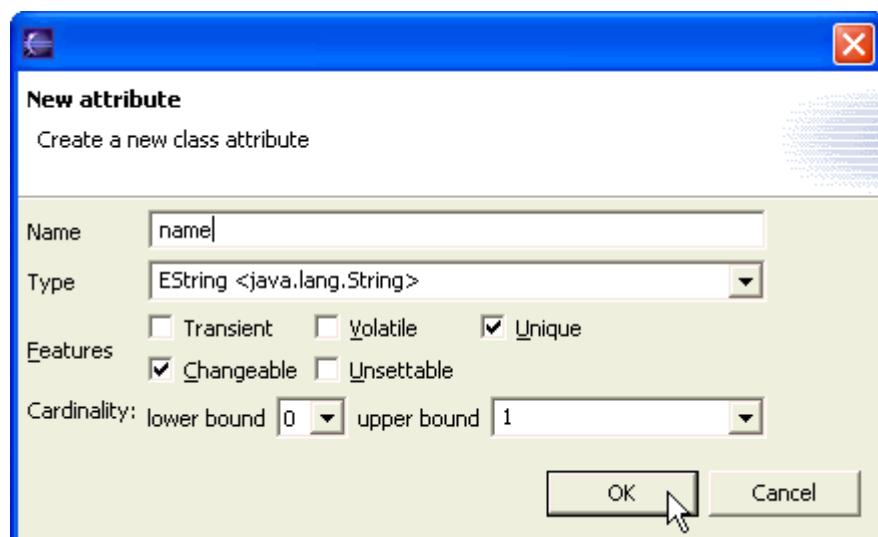


Attributes

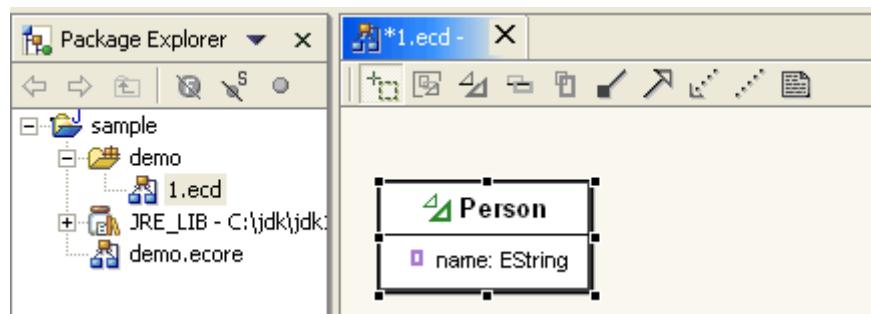
To create a new attribute, select *New > Attribute* in the EMF class contextual menu.



It launches an attribute creation dialog box. You have to set the attribute name, type, features and cardinality.

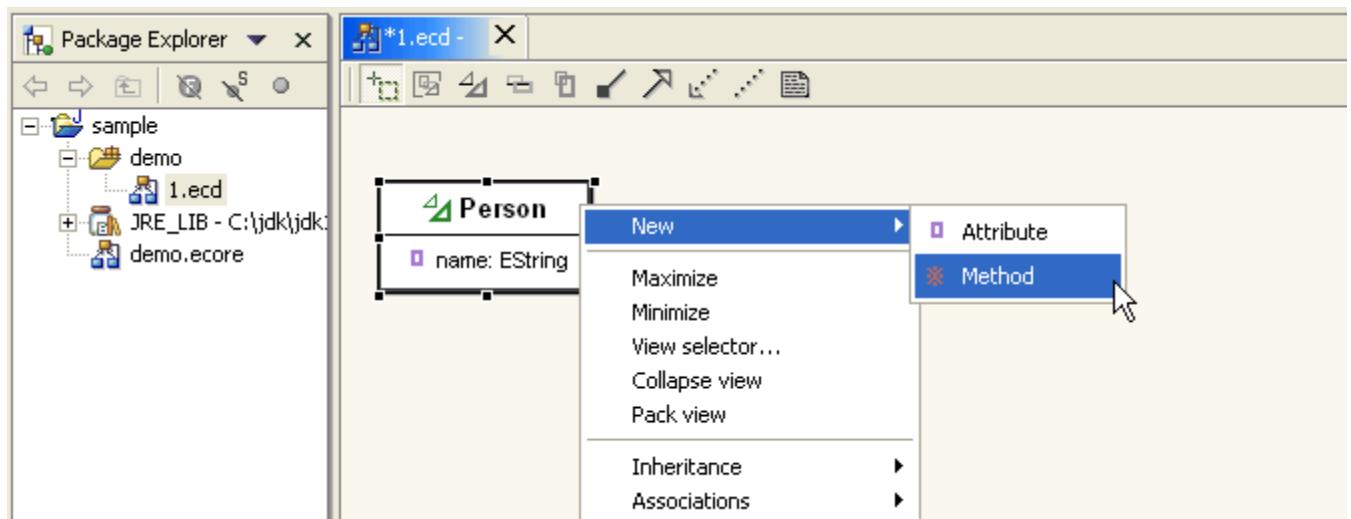


After validating the dialog box, the attribute appears immediatly in the class.

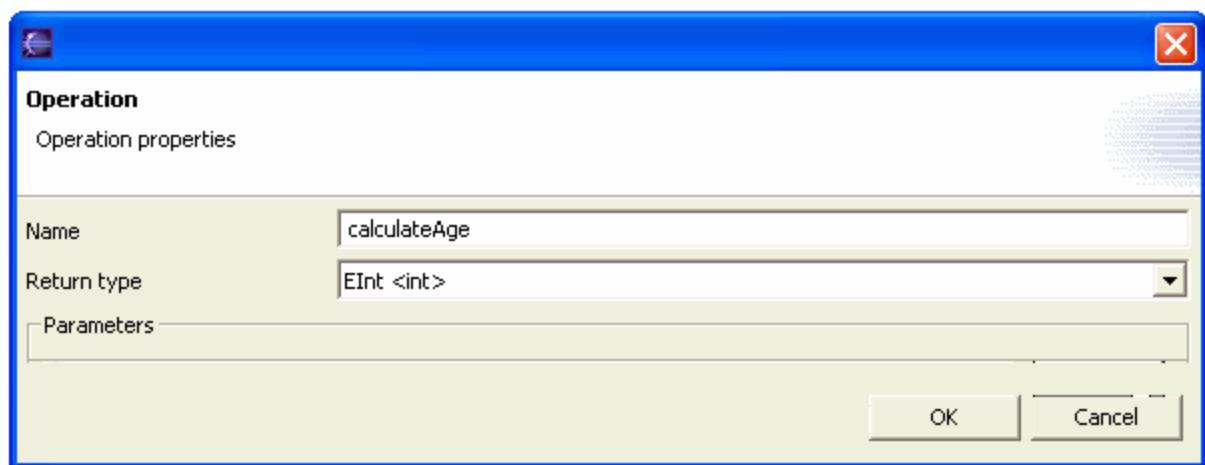


Methods

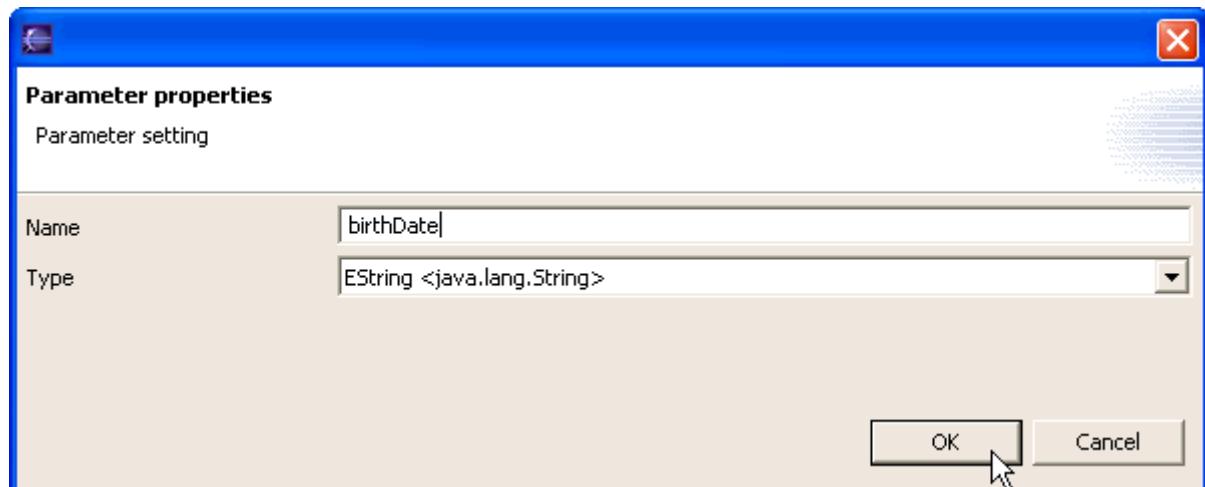
To create a new method, select *New > Method* in the EMF class contextual menu.



It launches a method creation dialog box. You have to set the method name, return type, parameters, and exceptions.



For each parameter, you must indicate its name and type.

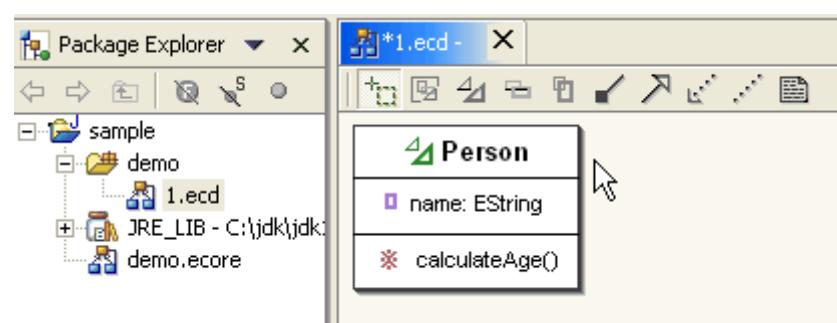


Operation

Operation properties

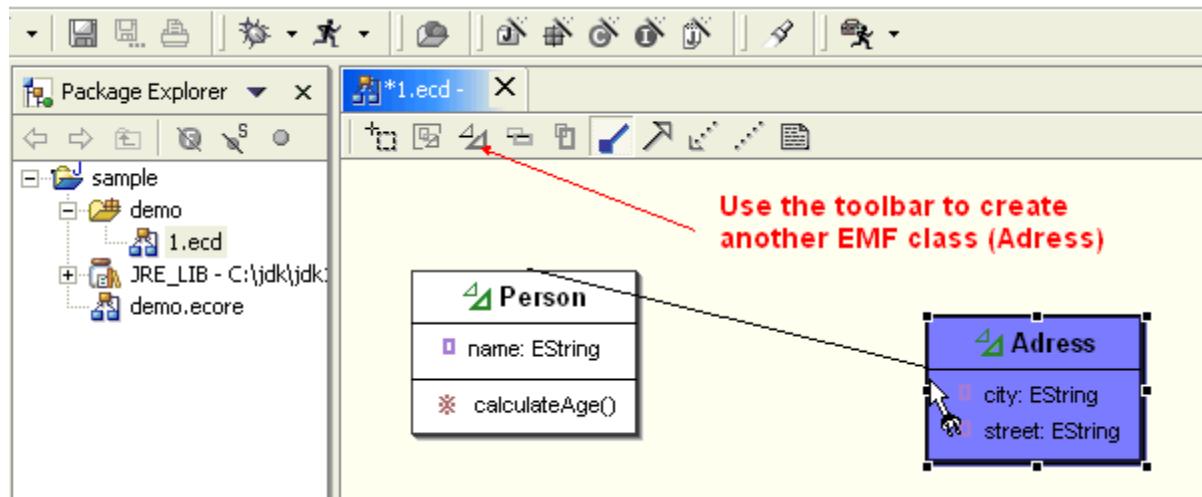
Name	calculateAge
Return type	EInt <int>
Parameters	
birthDate : EString	Add
	Remove

The method appears immediately in the class. If you click on the method, its code appears in the java editor area.

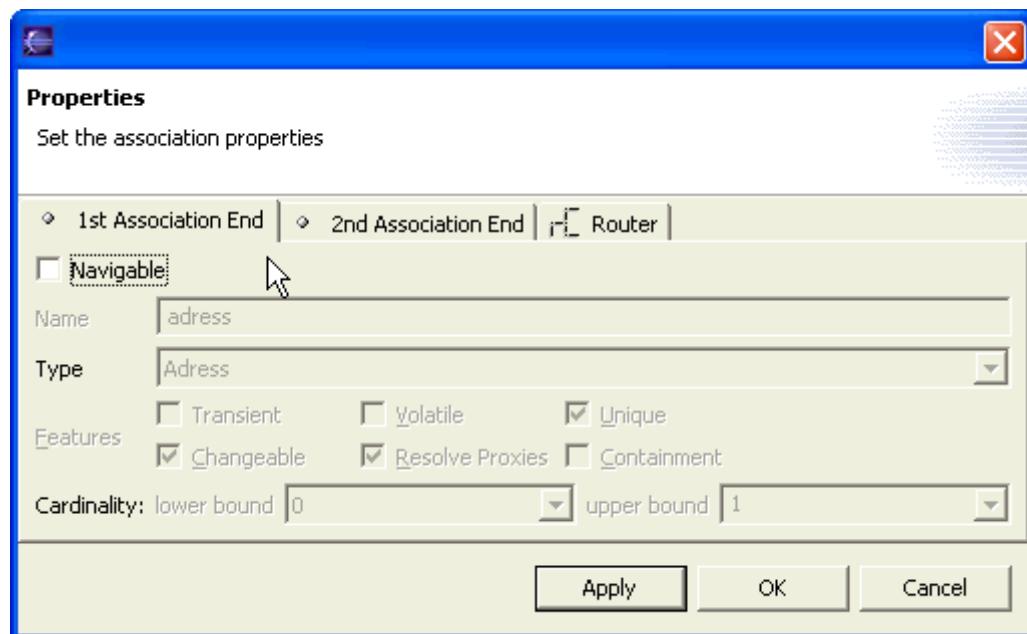


Associations

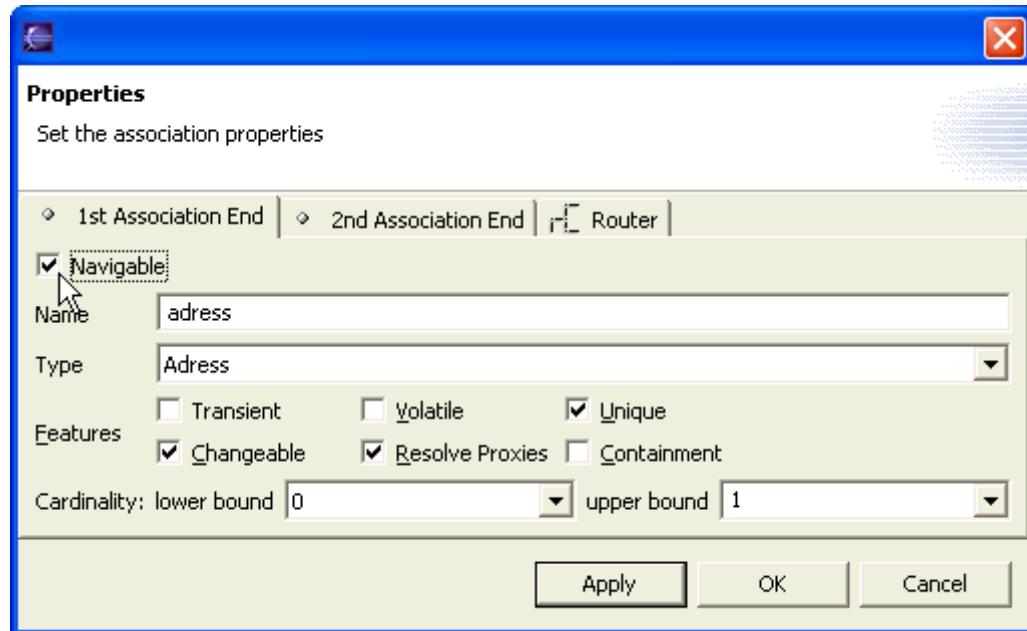
To create an association, select the *Association* toolbar button then click in the original class.



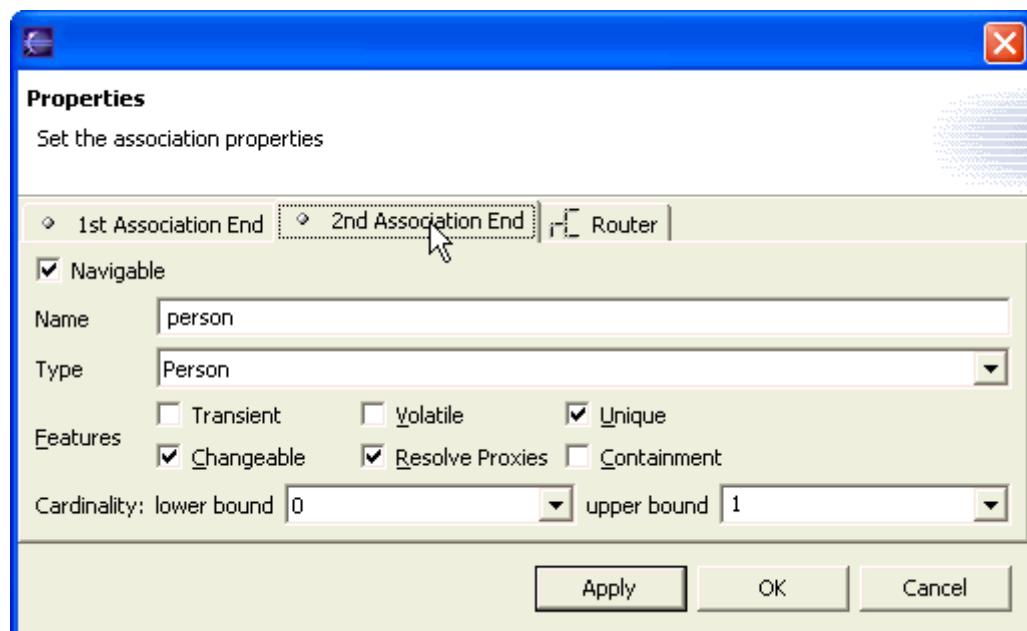
When you click in the target class, a dialog box appears.



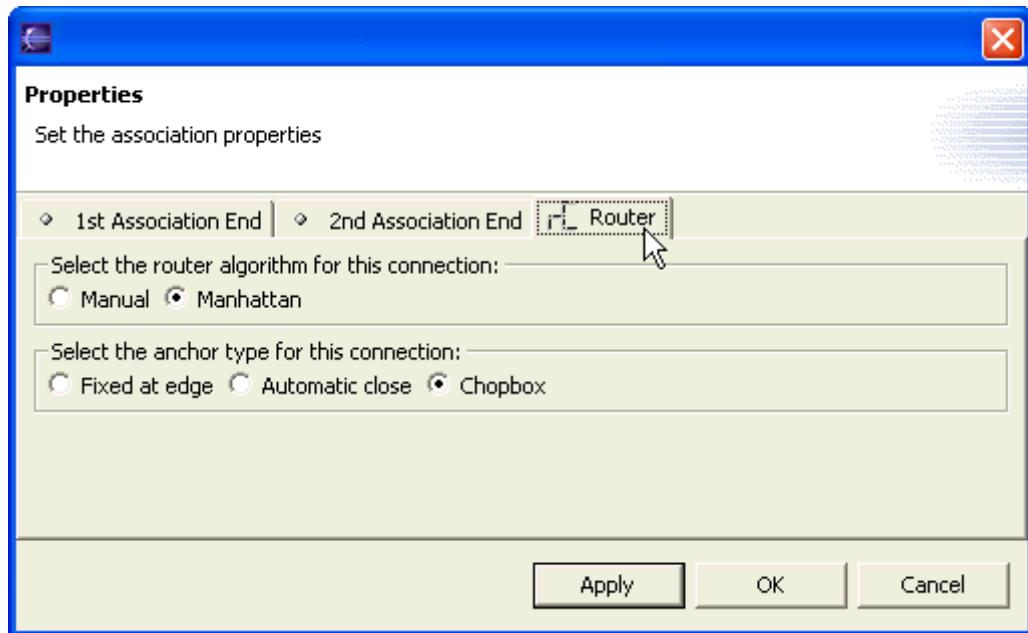
Use it to define the first association end...



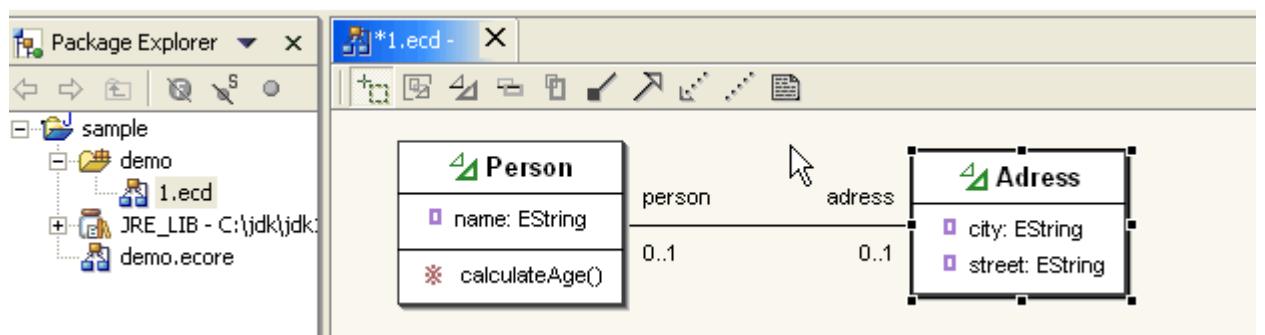
...then the second one. Remember, one of association end should be navigable.



And finally, the router algorithm and anchor strategy.

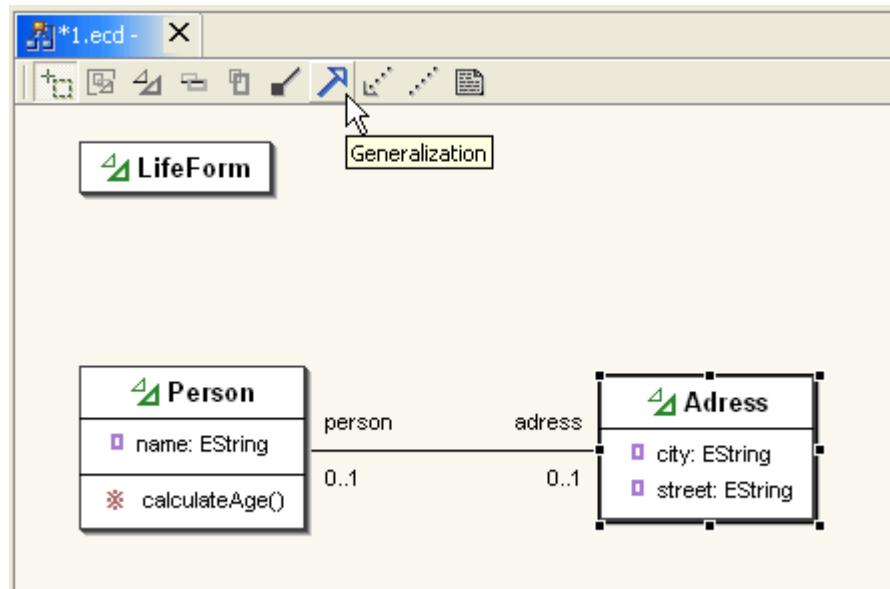


After validation, the association appears in the model.

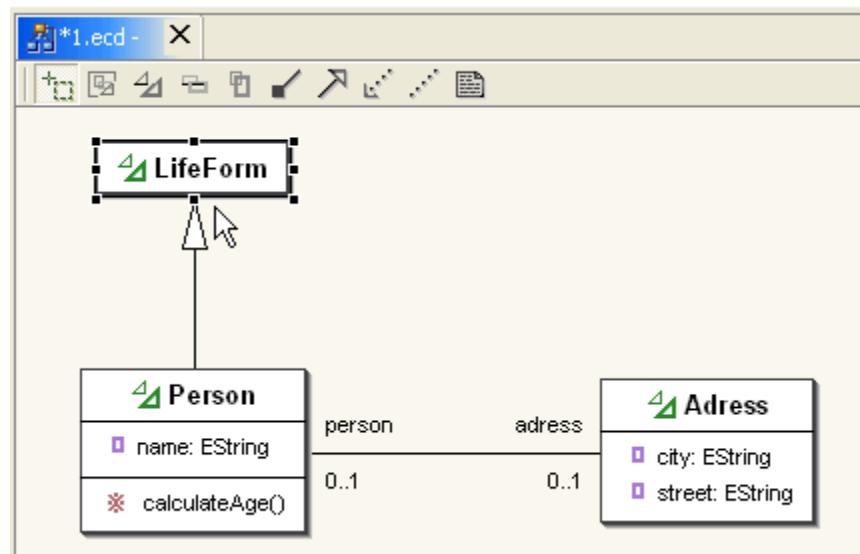


Implementation and Inheritance

To declare an implementation or an inheritance link, select the generalization toolbar button.



Select the source class or the source interface...



Eclipse Database

Overview

1. [Introduction](#)
2. [Modeling](#)
 1. [Database Connection](#)
 2. [Database Schema](#)
 3. [Database Schema DTD](#)
 4. [Database Diagram](#)
 5. [Database SQL](#)
 6. [Database Data](#)
 7. [Database Data DTD](#)
3. [Code Generation](#)
 1. [Torque](#)
 2. [Object Relational Bridge](#)
 3. [Hibernate](#)

1. Introduction

EclipseDatabase is a general purpose Database Tool.

It lets you manage the design of your Database and generate java code to access your Database.

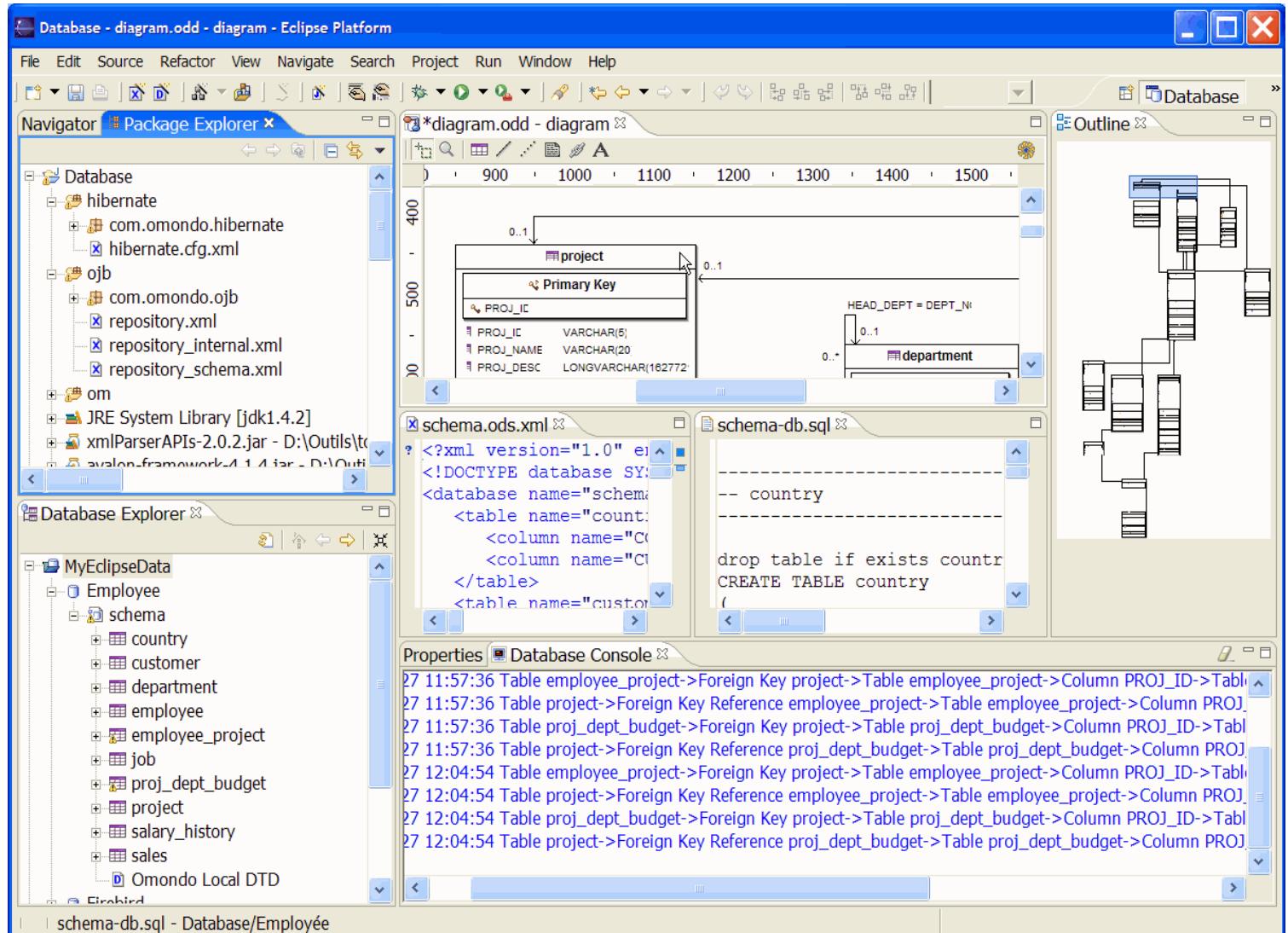
This tool is designed with open technologies which provide software independance.

- [Eclipse](#)
- [GEF](#)
- [Torque](#)
- [Village](#)
- [Hibernate](#)

Everything is stored in XML, there is no binary cryptic information.

Most of the generators are templates based and so users can modify the templates to fit their needs.

- Velocity



The tool is designed to address the most used Databases.

- Axion
- Cloudscape
- DB2
- DB2 400
- Hypersonic
- Informix
- Firebird
- Interbase
- Microsoft Access
- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL
- SapDB
- Sybase

To Access your Database, EclipseDatabase proposes you several Object Mapping technologies.

- Torque
- Object Relational Bridge
- Hibernate

2. Modeling

Database Modeling is Java independent.

It means that you can either create Database Connection and Diagrams into Simple Project, Java Project, C Project or other kind of Projects existing on your Eclipse Platform.

This tool focuses on database description and its associated Database Description Language SQL.

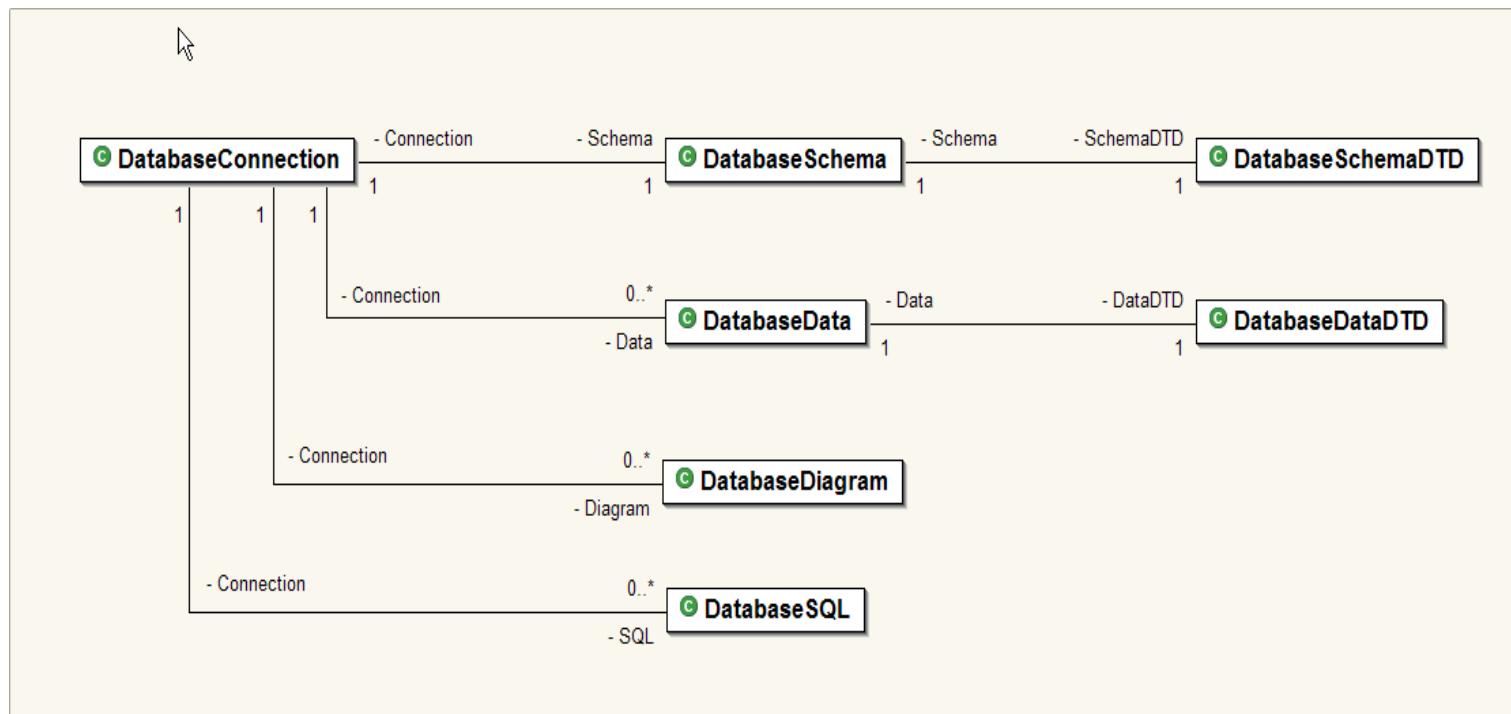
The goal of the tool is to provide a Database Modeler to Development Team.

This tool is not Database administrator or production targeted.

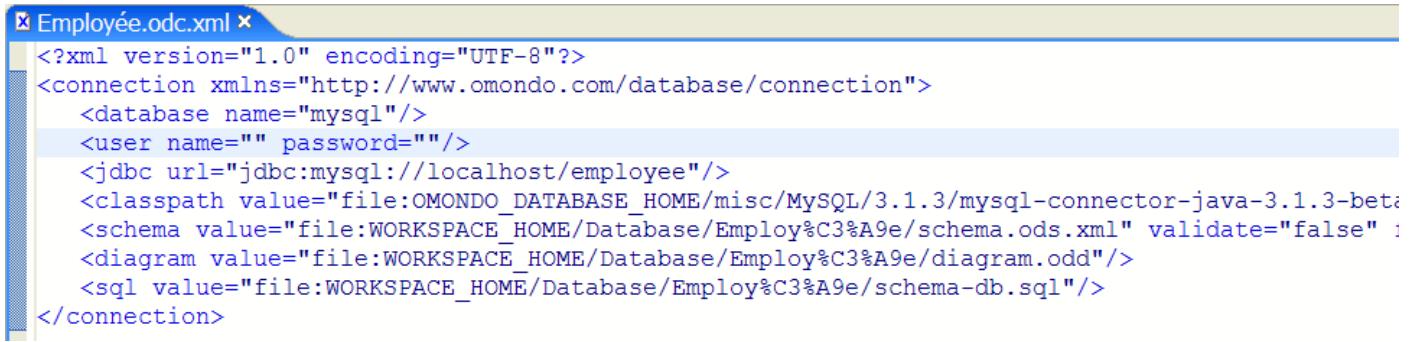
2.1. Database Connection

The Database Connection file is the central piece in EclipseDatabase.

The following UML diagrams show you how the different database files are connected.



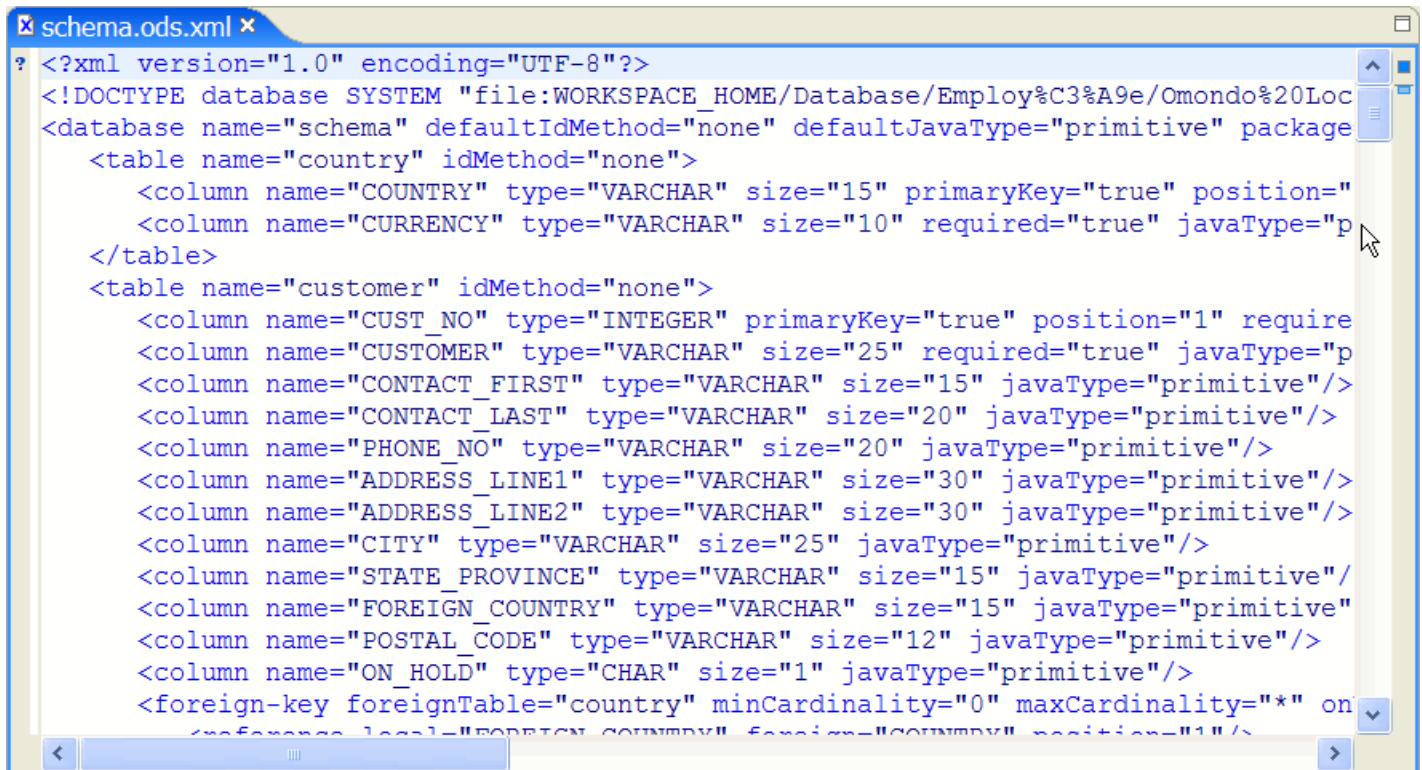
This xml file gathers the references for each database piece :



```
<?xml version="1.0" encoding="UTF-8"?>
<connection xmlns="http://www.omondo.com/database/connection">
  <database name="mysql"/>
  <user name="" password="" />
  <jdbc url="jdbc:mysql://localhost/employee"/>
  <classpath value="file:OMONDO_DATABASE_HOME/misc/MySQL/3.1.3/mysql-connector-java-3.1.3-beta.jar" />
  <schema value="file:WORKSPACE_HOME/Database/Employ%C3%A9/schema.ods.xml" validate="false" />
  <diagram value="file:WORKSPACE_HOME/Database/Employ%C3%A9/diagram.odd" />
  <sql value="file:WORKSPACE_HOME/Database/Employ%C3%A9/schema-db.sql" />
</connection>
```

- **Database Schema**, one Database Schema per connection
- **Database Schema DTD**, one Database Schema DTD per Database Schema
- **Database Diagram**, zero or * Database Diagrams per Database Connection
- **Database SQL**, zero or * Database SQL per Database Connection
- **Database Data**, zero or * Database Data per Database Connection
- **Database Data DTD**, one Database Data DTD per Database Data

2.2. Database Schema



```
? <?xml version="1.0" encoding="UTF-8"?>
!DOCTYPE database SYSTEM "file:WORKSPACE_HOME/Database/Employ%C3%A9/Omondo%20LocalSchema.ods"
<database name="schema" defaultIdMethod="none" defaultJavaType="primitive" package="com.emploi">
  <table name="country" idMethod="none">
    <column name="COUNTRY" type="VARCHAR" size="15" primaryKey="true" position="1" required="true" javaType="String" />
    <column name="CURRENCY" type="VARCHAR" size="10" required="true" javaType="String" />
  </table>
  <table name="customer" idMethod="none">
    <column name="CUST_NO" type="INTEGER" primaryKey="true" position="1" required="true" javaType="int" />
    <column name="CUSTOMER" type="VARCHAR" size="25" required="true" javaType="String" />
    <column name="CONTACT_FIRST" type="VARCHAR" size="15" javaType="primitive" />
    <column name="CONTACT_LAST" type="VARCHAR" size="20" javaType="primitive" />
    <column name="PHONE_NO" type="VARCHAR" size="20" javaType="primitive" />
    <column name="ADDRESS_LINE1" type="VARCHAR" size="30" javaType="primitive" />
    <column name="ADDRESS_LINE2" type="VARCHAR" size="30" javaType="primitive" />
    <column name="CITY" type="VARCHAR" size="25" javaType="primitive" />
    <column name="STATE_PROVINCE" type="VARCHAR" size="15" javaType="primitive" />
    <column name="FOREIGN_COUNTRY" type="VARCHAR" size="15" javaType="primitive" />
    <column name="POSTAL_CODE" type="VARCHAR" size="12" javaType="primitive" />
    <column name="ON_HOLD" type="CHAR" size="1" javaType="primitive" />
    <foreign-key foreignTable="country" minCardinality="0" maxCardinality="*" on="none" referenceLocal="FOREIGN_COUNTRY" foreign="COUNTRYID" position="1" />
  </table>
</database>
```

This xml file gathers all of your Database Meta Datas description.
 The Database Schemas are [Apache Torque](#) and [Apache Turbine](#) compatible.
 There is only one Schema associated to a Connection.

2.3. Database Schema DTD



```

Omondo Local DTD.dtd x
converted to lowercase.
javaname - sa Database/Employée/Omondo Local DTD.dtd's are converted
to lowercase.

-->

<!ELEMENT database (external-schema*, ((table|view)+))>
<!ATTLIST database
  name CDATA #IMPLIED
  defaultIdMethod (idbroker|native|none) "none"
  defaultJavaType (object|primitive) "primitive"
  package CDATA #IMPLIED
  baseClass CDATA #IMPLIED
  basePeer CDATA #IMPLIED
  defaultJavaNamingMethod (nochange|underscore|javaname) "underscore"
  defaultForeignKeyOnUpdate (cascade|setnull|restrict|none) "none"
  defaultForeignKeyonDelete (cascade|setnull|restrict|none) "none"
  heavyIndexing (true|false) "false"
>

<!ELEMENT external-schema EMPTY>
<!ATTLIST external-schema
  filename CDATA #REQUIRED

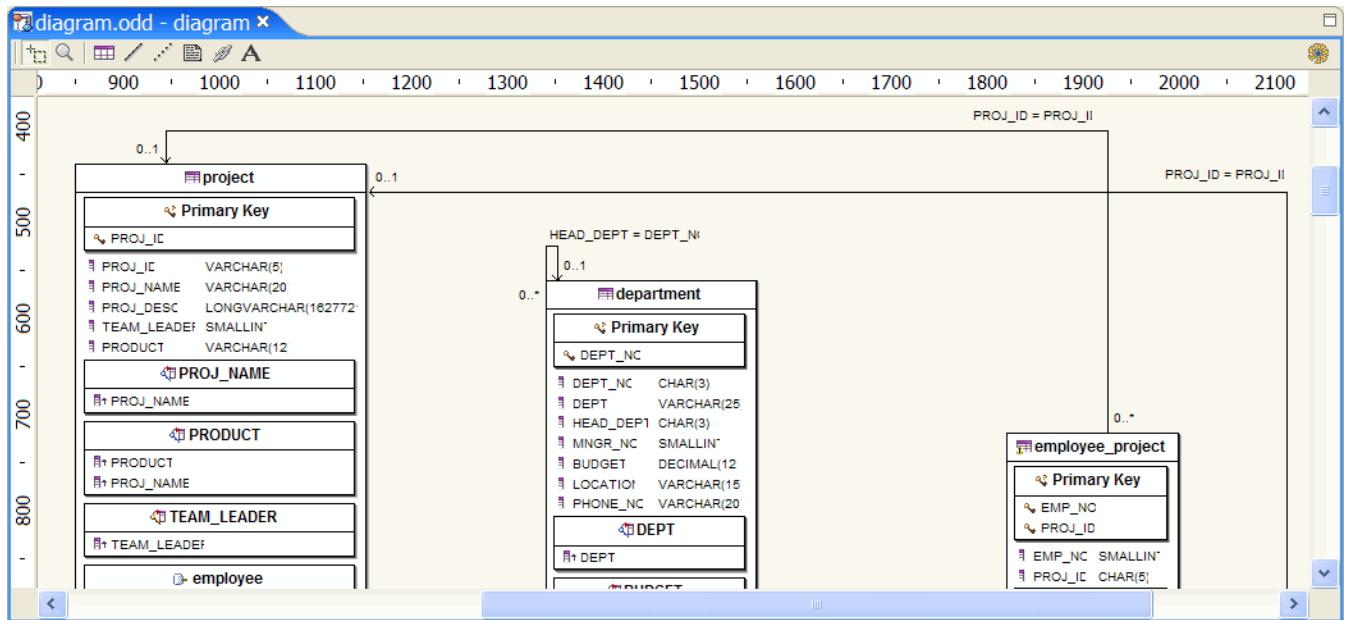
```

Each Schema file is associated with one DTD.

This DTD acts as a dictionary of your Database Schema Meta Data.

This DTD is [Apache Torque](#) and [Apache Turbine](#) compatible.

2.4. Database Diagram



Each Database Connection can view its Schemas in a graphical way.

This is the purpose of the Database Diagram.

You can have more than one diagram associated to a Database Connection.

This file is xml based :

```
<?xml version="1.0" encoding="UTF-8"?>
<editmodel:DatabaseDiagramEditModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <children xsi:type="editmodel:TableEditModel" targetConnections="//@children.5/@sourceConnection">
    <children xsi:type="editmodel:PrimaryKeyEditModel" size="137,59" id="country/%">
      <children xsi:type="editmodel:PrimaryKeyColumnEditModel" size="94,16" id="country/%/COUNTRY">
        </children>
      <children xsi:type="editmodel:ColumnEditModel" size="192,16" id="country/COUNTRY"/>
      <children xsi:type="editmodel:ColumnEditModel" size="202,16" id="country/CURRENCY"/>
    </children>
    <children xsi:type="editmodel:TableEditModel" targetConnections="//@children.9/@sourceConnection">
      <children xsi:type="editmodel:PrimaryKeyEditModel" size="137,59" id="customer/%">
        <children xsi:type="editmodel:PrimaryKeyColumnEditModel" size="92,16" id="customer/%/CUST_NO">
          </children>
        <children xsi:type="editmodel:ColumnEditModel" size="161,16" id="customer/CUST_NO"/>
        <children xsi:type="editmodel:ColumnEditModel" size="203,16" id="customer/CUSTOMER"/>
        <children xsi:type="editmodel:ColumnEditModel" size="233,16" id="customer/CONTACT_FIRST"/>
        <children xsi:type="editmodel:ColumnEditModel" size="229,16" id="customer/CONTACT_LAST"/>
        <children xsi:type="editmodel:ColumnEditModel" size="202,16" id="customer/PHONE_NO"/>
        <children xsi:type="editmodel:ColumnEditModel" size="235,16" id="customer/ADDRESS_LINE1"/>
        <children xsi:type="editmodel:ColumnEditModel" size="235,16" id="customer/ADDRESS_LINE2"/>
        <children xsi:type="editmodel:ColumnEditModel" size="158,16" id="customer/CITY"/>
        <children xsi:type="editmodel:ColumnEditModel" size="158,16" id="customer/STATE_PROVINCE"/>
      </children>
    </children>
  </children>
</editmodel:DatabaseDiagramEditModel>
```

2.5. Database SQL

The screenshot shows a window titled 'schema-db.sql' containing an SQL script. The script includes comments for 'country' and 'customer' tables, and creates them with specific columns and constraints. The code is as follows:

```

-- country
-----
drop table if exists country;
CREATE TABLE country
(
    COUNTRY VARCHAR(15) NOT NULL,
    CURRENCY VARCHAR(10) NOT NULL,
    PRIMARY KEY(COUNTRY)
) Type=InnoDB;

-- customer
-----

drop table if exists customer;
CREATE TABLE customer
(
    CUST_NO INTEGER default 0 NOT NULL,
    CUSTOMER VARCHAR(25) NOT NULL
)

```

The tool is able to generate two kinds of SQL files :

- Database Create SQL Script
- Database Schema SQL Script

2.6. Database Data

The screenshot shows a window titled 'data-db.odx.xml' containing an XML dataset. The XML structure represents employees and their departments. The code is as follows:

```

? <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dataset SYSTEM "file:WORKSPACE_HOME/Database/Oracle/data-db.dtd">
<dataset name="all">
    <Dept Deptno="10" Dname="ACCOUNTING" Loc="NEW YORK"/>
    <Dept Deptno="20" Dname="RESEARCH" Loc="DALLAS"/>
    <Dept Deptno="30" Dname="SALES" Loc="CHICAGO"/>
    <Dept Deptno="40" Dname="OPERATIONS" Loc="BOSTON"/>
    <Emp Empno="7369" Ename="SMITH" Job="CLERK" Mgr="7902" Hiredate="1980-12-17" Sal="800" Deptno="20"/>
    <Emp Empno="7499" Ename="ALLEN" Job="SALESMAN" Mgr="7698" Hiredate="1981-02-20" Sal="1600" Comm="300" Deptno="30"
    <Emp Empno="7521" Ename="WARD" Job="SALESMAN" Mgr="7698" Hiredate="1981-02-22" Sal="1250" Comm="500" Deptno="30"
    <Emp Empno="7566" Ename="JONES" Job="MANAGER" Mgr="7839" Hiredate="1981-04-02" Sal="2975" Deptno="20"/>
    <Emp Empno="7654" Ename="MARTIN" Job="SALESMAN" Mgr="7698" Hiredate="1981-09-28" Sal="1250" Comm="1400" Deptno='
    <Emp Empno="7698" Ename="BLAKE" Job="MANAGER" Mgr="7839" Hiredate="1981-05-01" Sal="2850" Deptno="30"/>
    <Emp Empno="7782" Ename="CLARK" Job="MANAGER" Mgr="7839" Hiredate="1981-06-09" Sal="2450" Deptno="10"/>
    <Emp Empno="7788" Ename="SCOTT" Job="ANALYST" Mgr="7566" Hiredate="1987-04-19" Sal="3000" Deptno="20"/>
    <Emp Empno="7839" Ename="KING" Job="PRESIDENT" Hiredate="1981-11-17" Sal="5000" Deptno="10"/>
    <Emp Empno="7844" Ename="TURNER" Job="SALESMAN" Mgr="7698" Hiredate="1981-09-08" Sal="1500" Comm="0" Deptno="30"
    <Emp Empno="7876" Ename="ADAMS" Job="CLERK" Mgr="7788" Hiredate="1987-05-23" Sal="1100" Deptno="20"/>
    <Emp Empno="7900" Ename="JAMES" Job="CLERK" Mgr="7698" Hiredate="1981-12-03" Sal="950" Deptno="30"/>
    <Emp Empno="7902" Ename="FORD" Job="ANALYST" Mgr="7566" Hiredate="1981-12-03" Sal="3000" Deptno="20"/>
    <Emp Empno="7934" Ename="MILLER" Job="CLERK" Mgr="7782" Hiredate="1982-01-23" Sal="1300" Deptno="10"/>
    <Salgrade Grade="1" Losal="700" Hisal="1200"/>
    <Salgrade Grade="2" Losal="1201" Hisal="1400"/>
    <Salgrade Grade="3" Losal="1401" Hisal="2000"/>
    <Salgrade Grade="4" Losal="2001" Hisal="3000"/>
    <Salgrade Grade="5" Losal="3001" Hisal="9999"/>
</dataset>

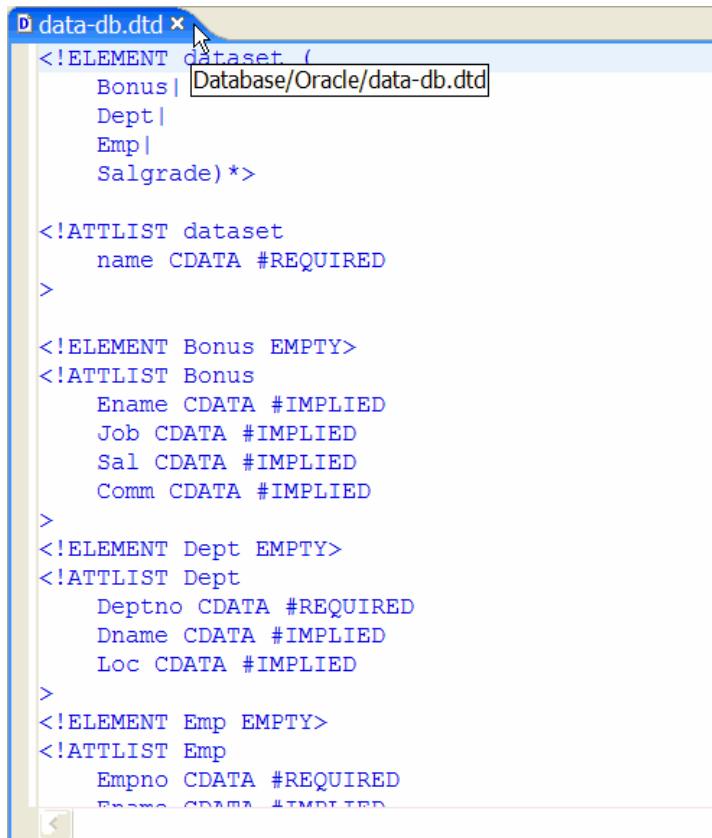
```

This xml file gathers all of your Database Data

This file is Database Connection specific.

This file is generated when running the [Database Data DTD and XML Resources](#) wizard.

2.7. Database Data DTD



```

data-db.dtd
<!ELEMENT dataset (
    Bonus |
    Dept |
    Emp |
    Salgrade)*>

<!ATTLIST dataset
    name CDATA #REQUIRED
>

<!ELEMENT Bonus EMPTY>
<!ATTLIST Bonus
    Ename CDATA #IMPLIED
    Job CDATA #IMPLIED
    Sal CDATA #IMPLIED
    Comm CDATA #IMPLIED
>
<!ELEMENT Dept EMPTY>
<!ATTLIST Dept
    Deptno CDATA #REQUIRED
    Dname CDATA #IMPLIED
    Loc CDATA #IMPLIED
>
<!ELEMENT Emp EMPTY>
<!ATTLIST Emp
    Empno CDATA #REQUIRED
    Empname CDATA #IMPLIED
>

```

This dtd file acts as the dictionary of a Database Data file.

This file is Database Connection specific.

This file is generated when running the [Database Data DTD and XML Resources](#) wizard.

3. Code Generation

Database Code generation is Java dependent.

It means that you need to create a Java Project to generate your Java Code.

However you can define a Simple Project to hold your Database description while you use a Java Project for your application.

3.1. Torque

Torque is a persistence layer.

Torque is an [Apache Software Foundation](#) Project.

EclipseDatabase is able to generate :

- **Torque.properties** file
- **Object Mapping** Java objects

3.2. Object Relational Bridge

[ObJect Relational Bridge \(OJB\)](#) is an Object/Relational mapping tool that allows transparent persistence for Java Objects against relational databases.

OJB is an [Apache Software Foundation](#) Project.

EclipseDatabase is able to generate :

- **repository.xml** file
- **repository_internal.xml** file
- **repository_schema** file
- **OJB** Java objects

3.3. Hibernate

[Hibernate](#) is an object/relational persistence and query service for Java.

EclipseDatabase is able to generate :

- **Hibernate configuration** file
- **Hibernate** Java objects
- **Stateless Session Bean**

Global Preferences

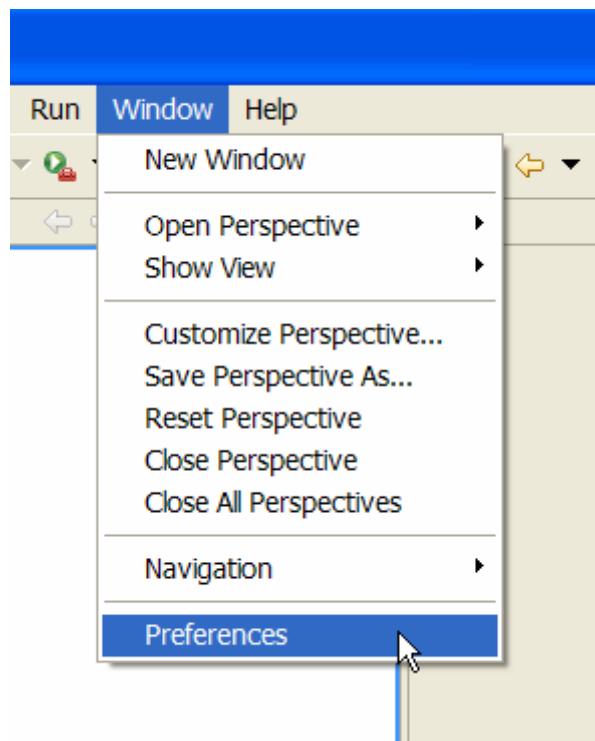
1. [Introduction](#)
2. [Database](#)
 1. [MyEclipseData](#)
 2. [Path Policy](#)
 3. [Edition Policy](#)
 4. [Save Policy](#)
3. [Database Connection](#)
4. [Database Diagram](#)
5. [Database Schema](#)
 1. [Database Schema DTD](#)
6. [Database SQL](#)
7. [Templates](#)

1. Introduction

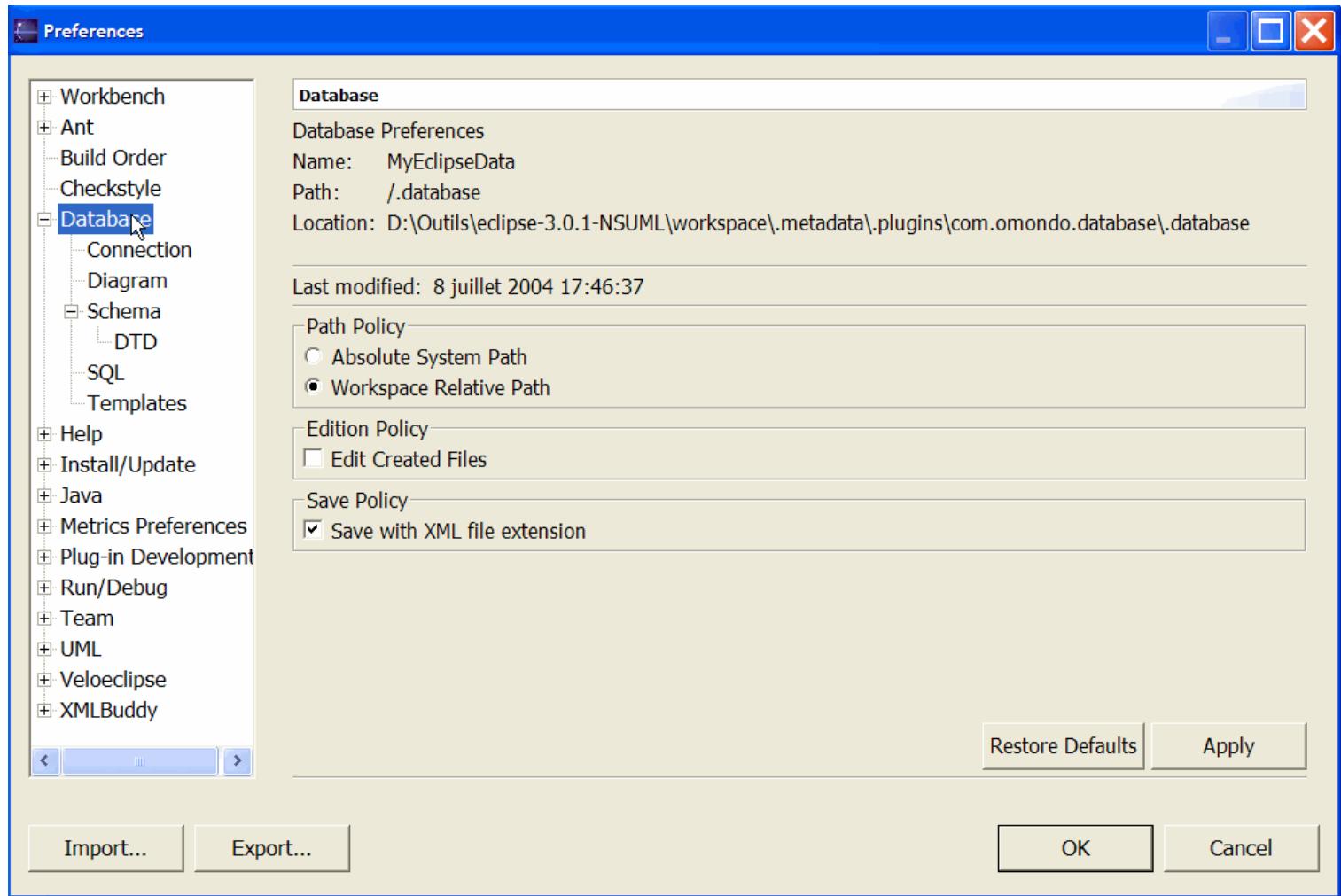
The Global Preferences allows the developer to set global settings.

2. Database

In the window menu select **Window -> Preferences**



Select the Database preference.



2.1. MyEclipseData

The MyEclipseData area is the physical directory where the MyEclipseData information is stored.

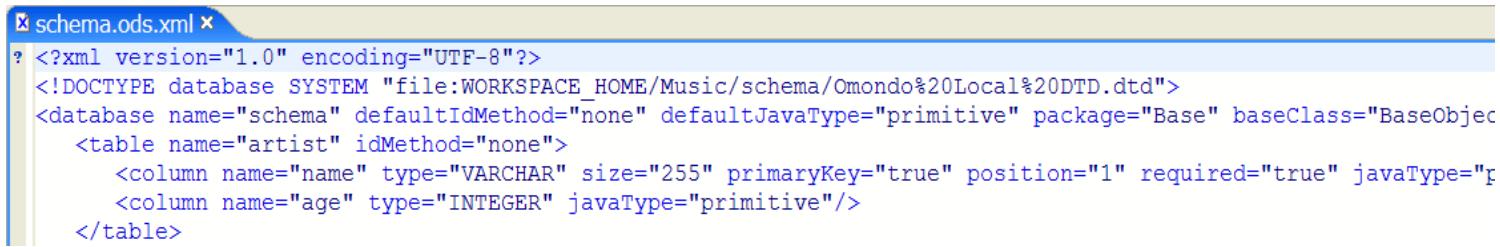
2.2. Path Policy

Each time you save a Schema file, a local DTD can be associated and created.

The path policy can either be generated as an Absolute System Path file or as a Workspace Relative Path.

```
schema.ods.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE database SYSTEM "file:D:/Outils/eclipse-3.0.1-NSUML/runtime-workspace/Music/schema/Omondo%20Local%20
<database name="schema" defaultIdMethod="none" defaultJavaType="primitive" package="Base" baseClass="BaseObject
  <table name="artist" idMethod="none">
    <column name="name" type="VARCHAR" size="255" primaryKey="true" position="1" required="true" javaType="p
      <column name="age" type="INTEGER" javaType="primitive"/>
  </table>
```

If you choose a Workspace Relative Path, the special WORKSPACE_HOME variable is used. This variable will be substituted to the real and absolute path of your Eclipse platform at runtime.



```
? <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE database SYSTEM "file:WORKSPACE_HOME/Music/schema/Omondo%20Local%20DTD.dtd">
<database name="schema" defaultIdMethod="none" defaultJavaType="primitive" package="Base" baseClass="BaseObjec
<table name="artist" idMethod="none">
    <column name="name" type="VARCHAR" size="255" primaryKey="true" position="1" required="true" javaType="p
        <column name="age" type="INTEGER" javaType="primitive"/>
    </table>
```

This feature is useful if you team work your diagrams.

Each developer usually installs Eclipse platform where he or she wants: one can install it on a Windows logical C: partition while another uses a Linux /usr/local/eclipse directory.

By using this variable, the set of database files will be easily shared with your team.

However, the use of this WORKSPACE_HOME variable is not XML normalized.

This means that some manual adjustments need to be done if it is used outside its EclipseDatabase scope.

2.3. Edition Policy

This feature allows Eclipse to automatically open a file with its default editor when a Database file is created.

2.4. Save Policy

You can control the way your XML file extensions are created using this option.

Connection	.odc.xml	.odc
Schema	.ods.xml	.ods
Data	.odx.xml	.odx
Diagram	.odd.xml	.odd
Hibernate	.hbm.xml	.hbm

This feature is useful if you want to give an XML behaviour to your Database file.

For example, as an XML file it will be automatically associated to all the XML editors existing in your Eclipse platform, otherwise you need to associate them explicitly through :

Window -> Preferences -> Workbench -> File Associations

Database Connection

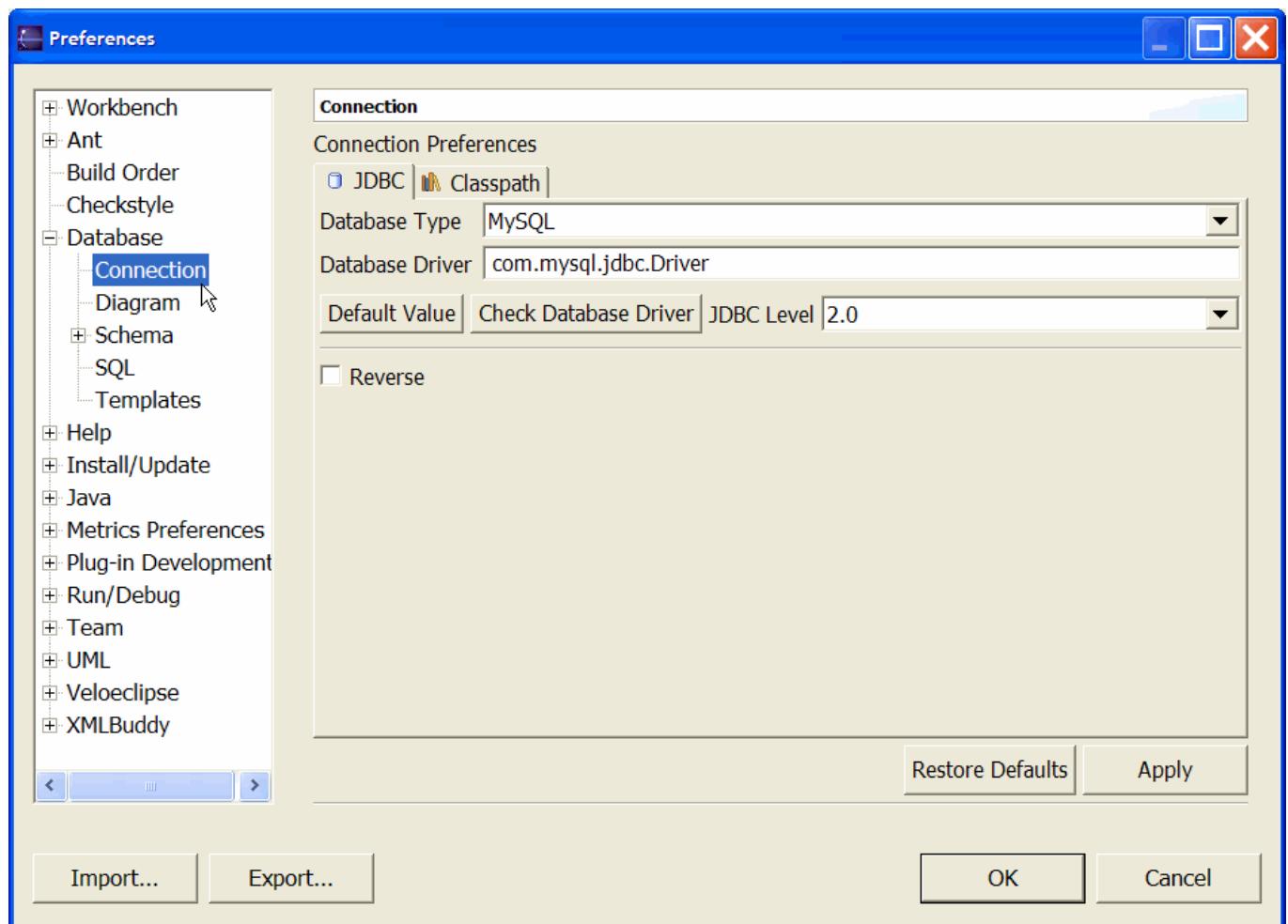
1. Introduction
2. JDBC
 1. Database Type
 2. Database Driver
 3. Default Value
 4. Check Database Driver
 5. JDBC Level
 6. Reverse
3. Classpath
 1. Add Jar
 2. Add Folder
 3. Add Variable
 1. OMONDO_DATABASE_HOME
 2. Bundled JDBC Drivers
 4. Edit Variable
 5. Up - Down
 6. Remove

1. Introduction

The Connection Preferences allows the developer to set Connection global settings.

These settings will be inherited to each new Database Connection wizard page.

2. JDBC



2.1. Database Type

The Database Type selector allows you to choose Database predefined drivers.

The Database Type selection sets a predefined Database Driver and a JDBC Level.

2.2. Database Driver

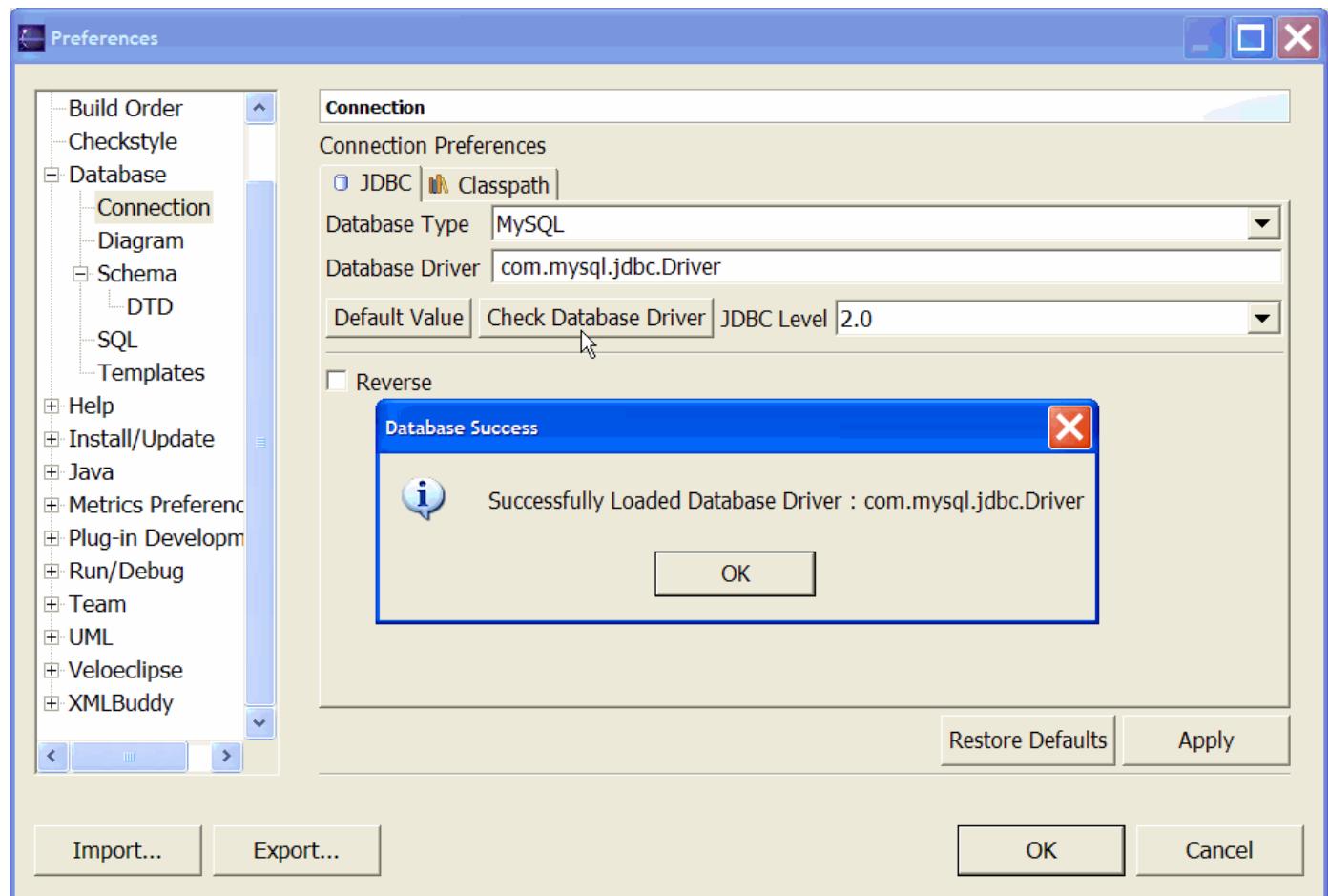
The Database Driver is an updatable field, this allows any modification to fit your particular needs.

2.3. Default Value

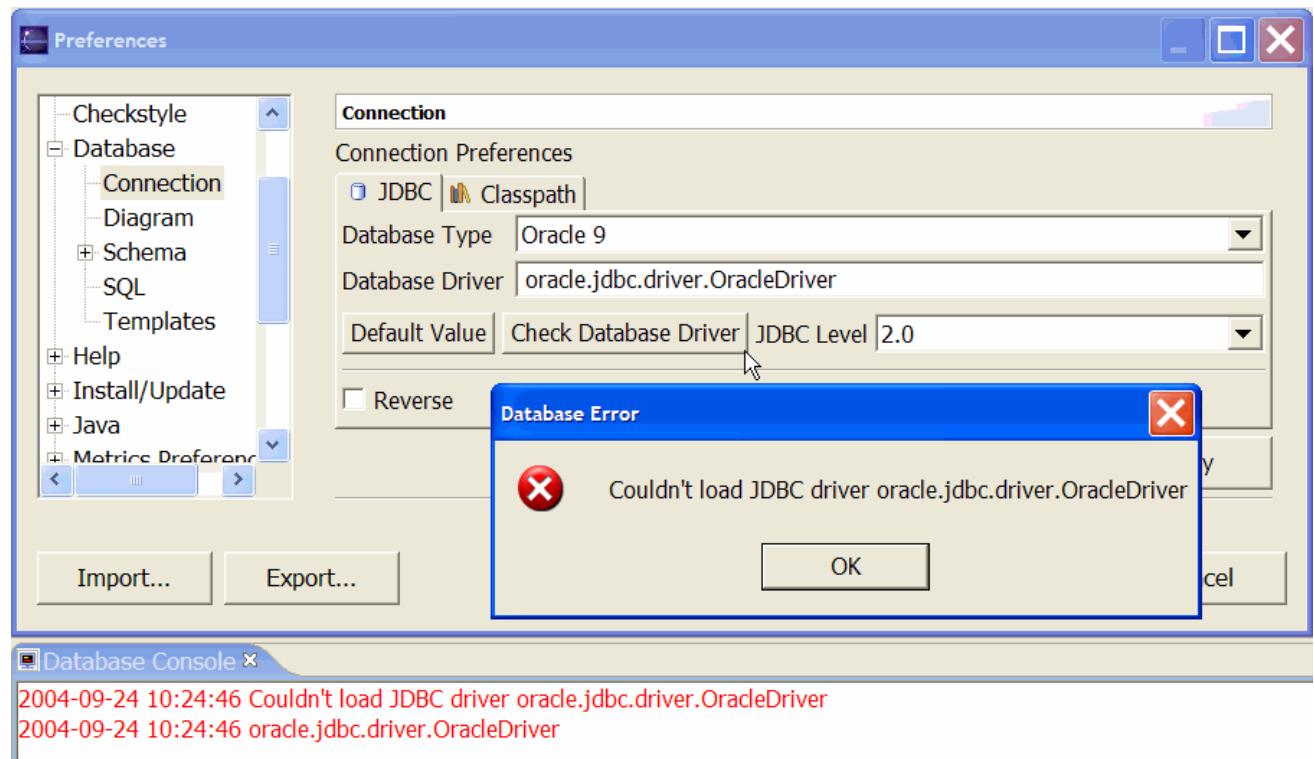
The Default value reset the Database Driver to its predefined value.

2.4. Check Database Driver

The Check Database Driver button tests the Eclipse Platform capability to load this particular JDBC Driver.



In the case of a Check failure, the [DatabaseConsole](#) gives you a short explanation about the failure reason :



2.5. JDBC Level

The JDBC Level allows you to choose a JDBC predefined Level.

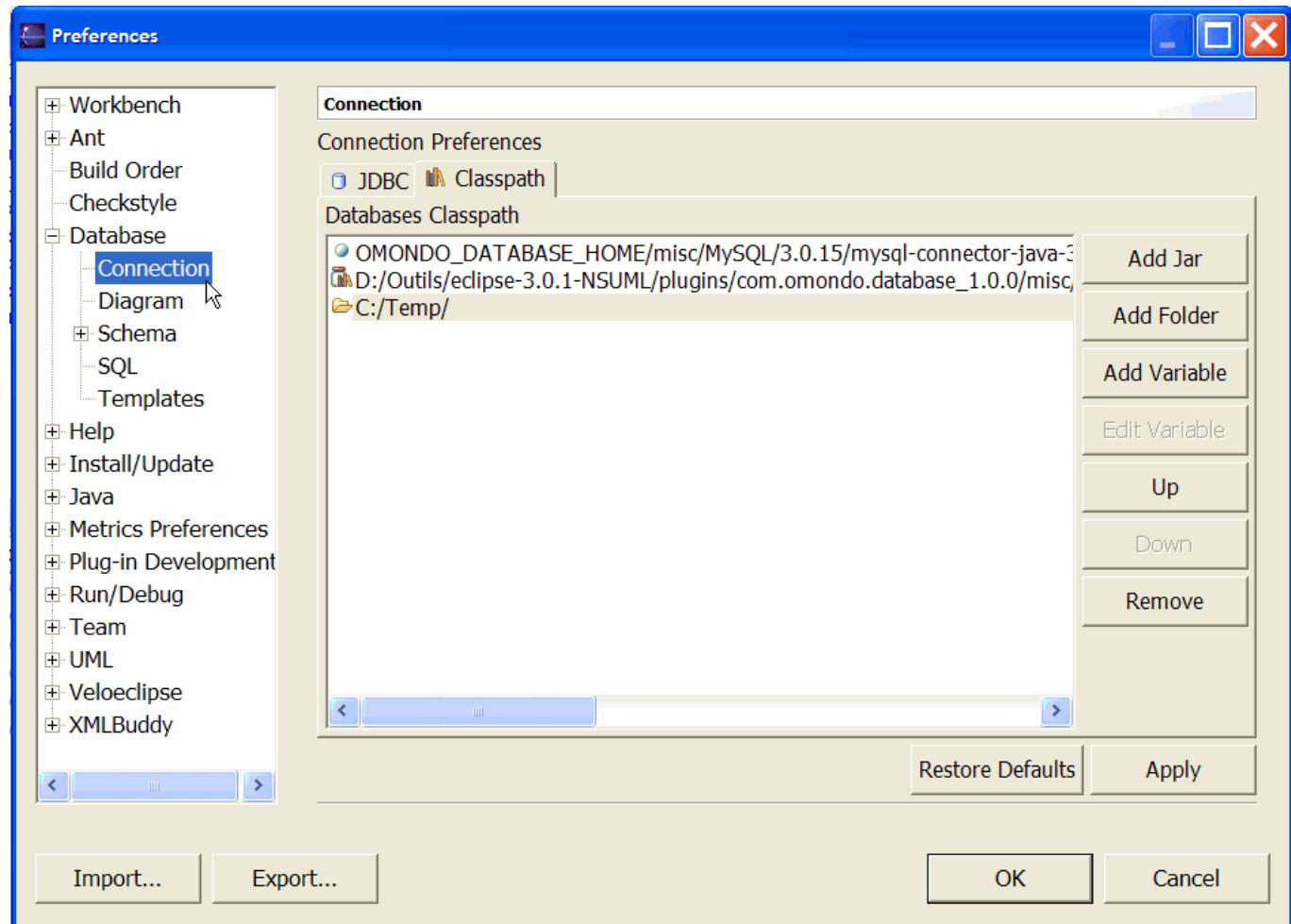
This value is used with the [OJB Code Generator](#).

2.6. Reverse

Each new Database Connection will attempt to reverse the Meta Data of a particular Database with the Reverse checked box.

3. Classpath

With this preference you can define where your JDBC drivers are stored.



3.1. Add Jar

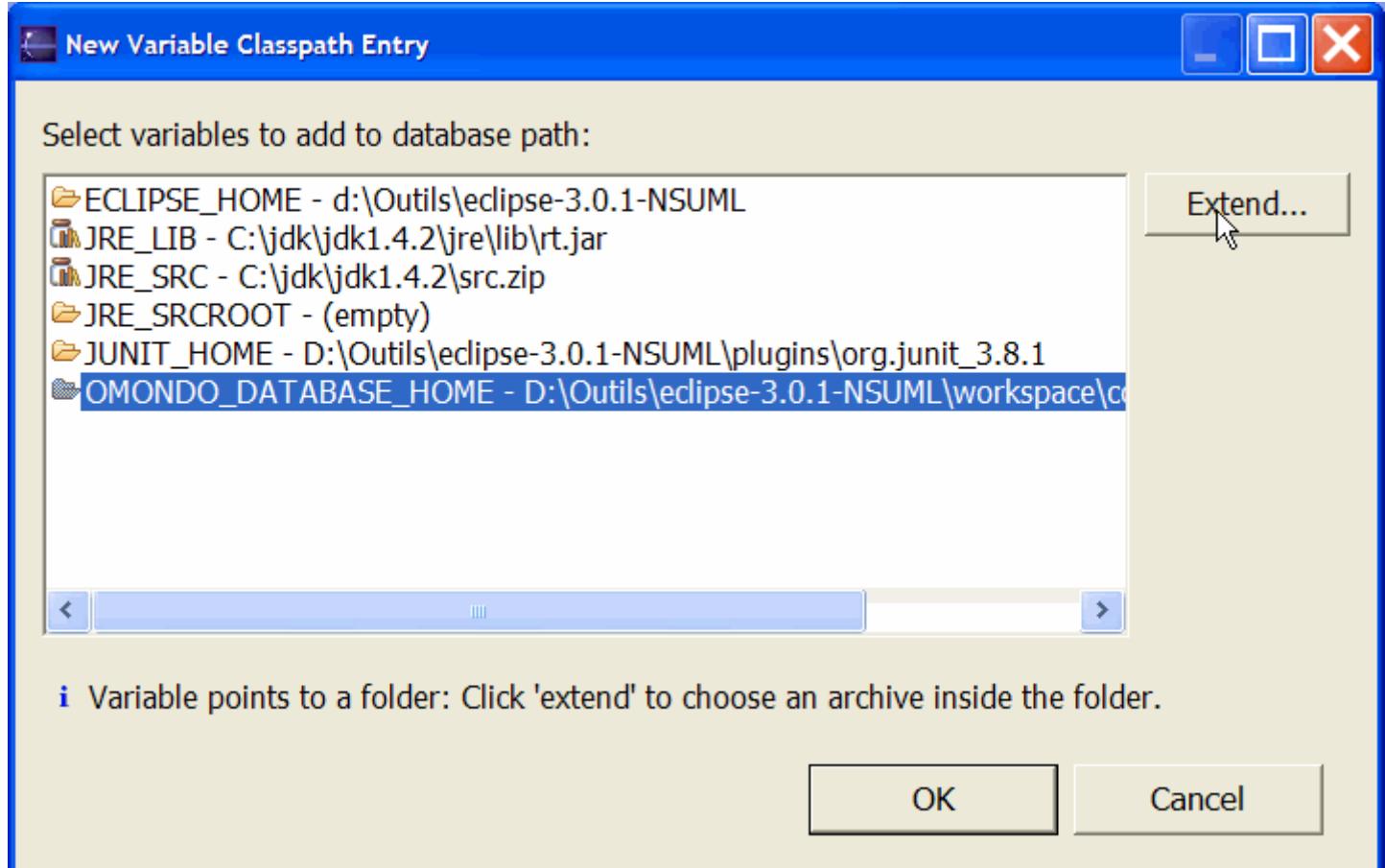
The Add Jar button opens a *.jar or *.zip selector.

3.2. Add Folder

The Add Folder button opens a folder selector.

3.3. Add Variable

With the Add Variable you can choose among the predefined Eclipse Platform JDT Variables.

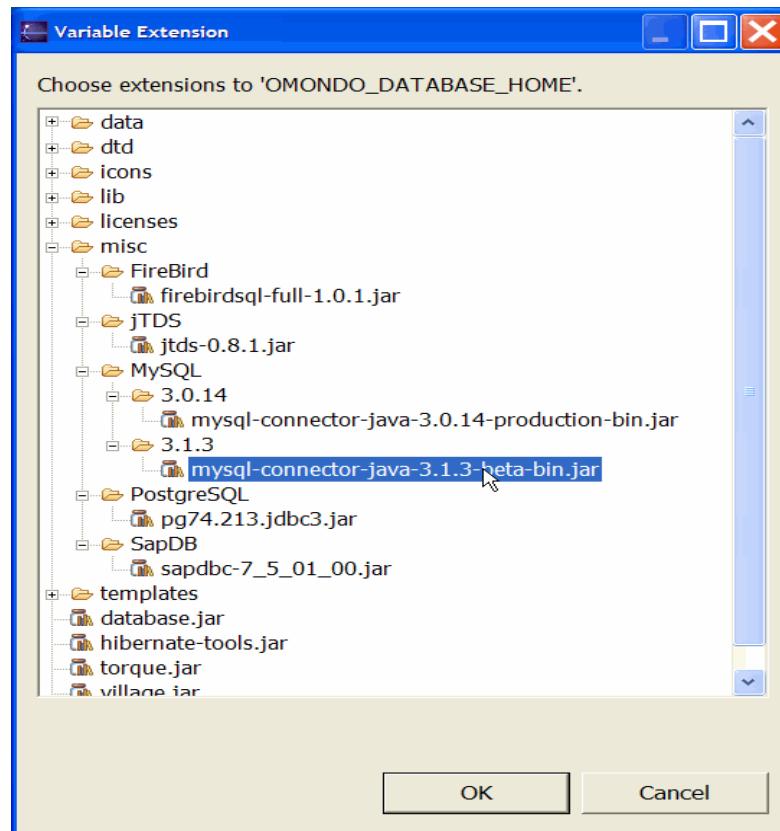


3.3.1. OMONDO_DATABASE_HOME

The OMONDO_DATABASE_HOME is pre-set to the EclipseDatabase root plugin.

The EclipseDatabase plugin has some known JDBC Drivers.

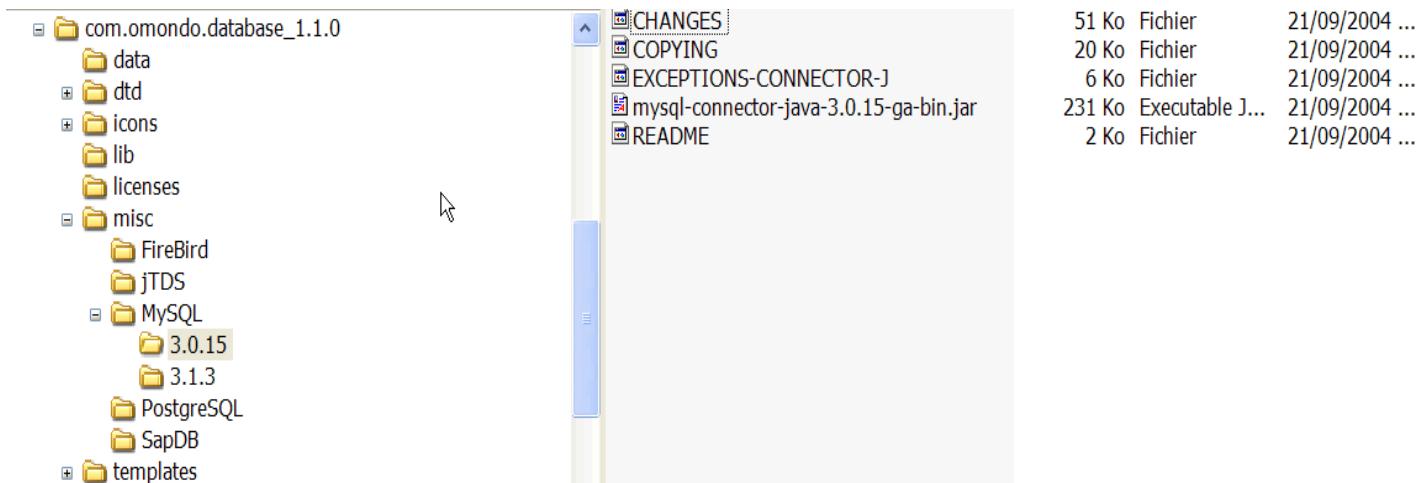
Extend the OMONDO_DATABASE_HOME variable :



3.3.2. Bundled JDBC Drivers

At the local file system level you can check the contents of each drivers directory.

You will see some additional files like the changelogs or the licence are provided :



The use of this variable has a consequence in team work.

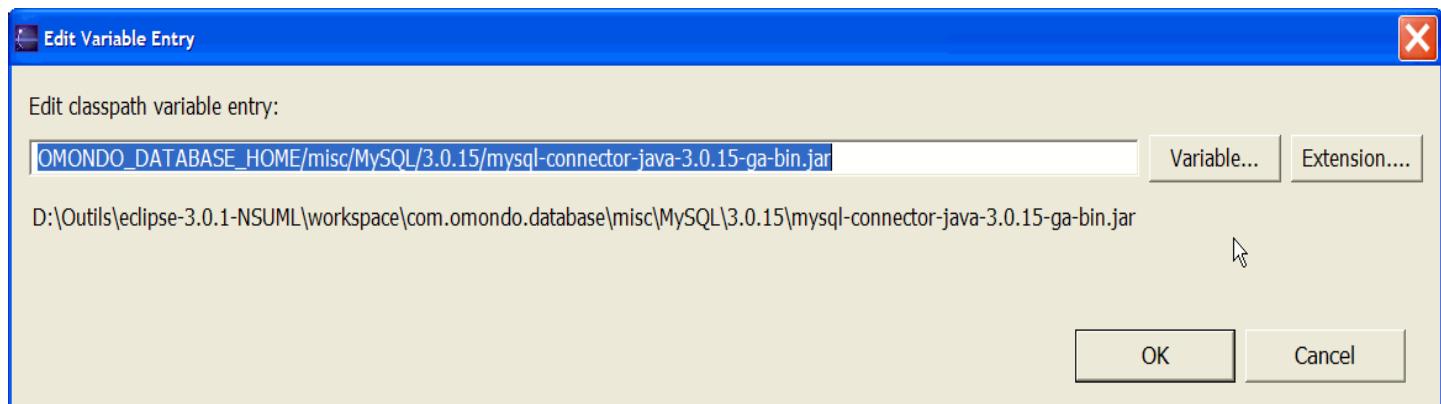
The connection file will store this variable.

If you share your Database files with a team system, by using the OMONDO_DATABASE_HOME variable set to the predefined drivers, your connection file will be platform independent.

```
Music.odc.xml
<?xml version="1.0" encoding="UTF-8"?>
<connection xmlns="http://www.omondo.com/database/connection">
    <database name="mysql"/>
    <user name="" password=""/>
    <jdbc url="jdbc:mysql://localhost/music"/>
    <classpath value="file:OMONDO_DATABASE_HOME/misc/MySQL/mysql-connector-java-3.0.14-production-bin.jar"/>
    <schema value="file:WORKSPACE_HOME/Music/schema/schema.ods.xml" validate="false" filterView="true" filterIn=>
    <diagram value="file:WORKSPACE_HOME/Music/schema/create/diagram.odd"/>
    <sql value="file:WORKSPACE_HOME/Music/schema/schema-db.sql"/>
</connection>
```

3.4. Edit Variable

With the Edit Variable button you can modify a variable content.



3.5. Up - Down

The Up or Down button lets you organize your classloader sequence.

3.6. Remove

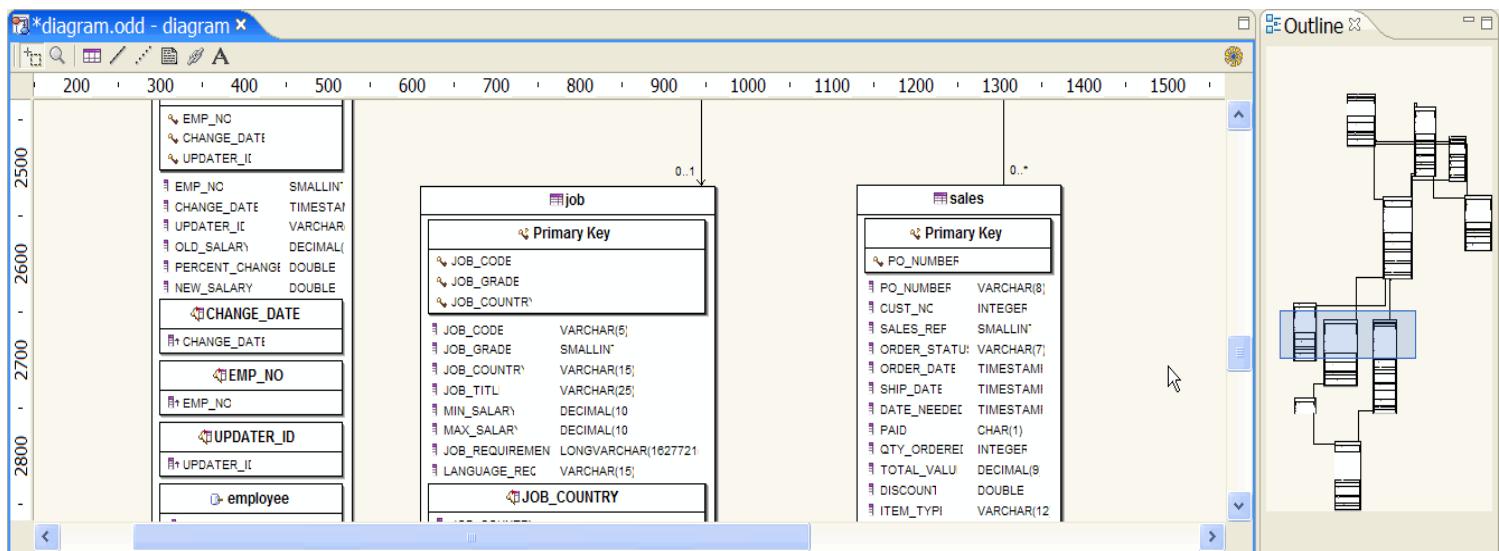
With the Remove button, you can delete a classpath entry.

Database Diagram

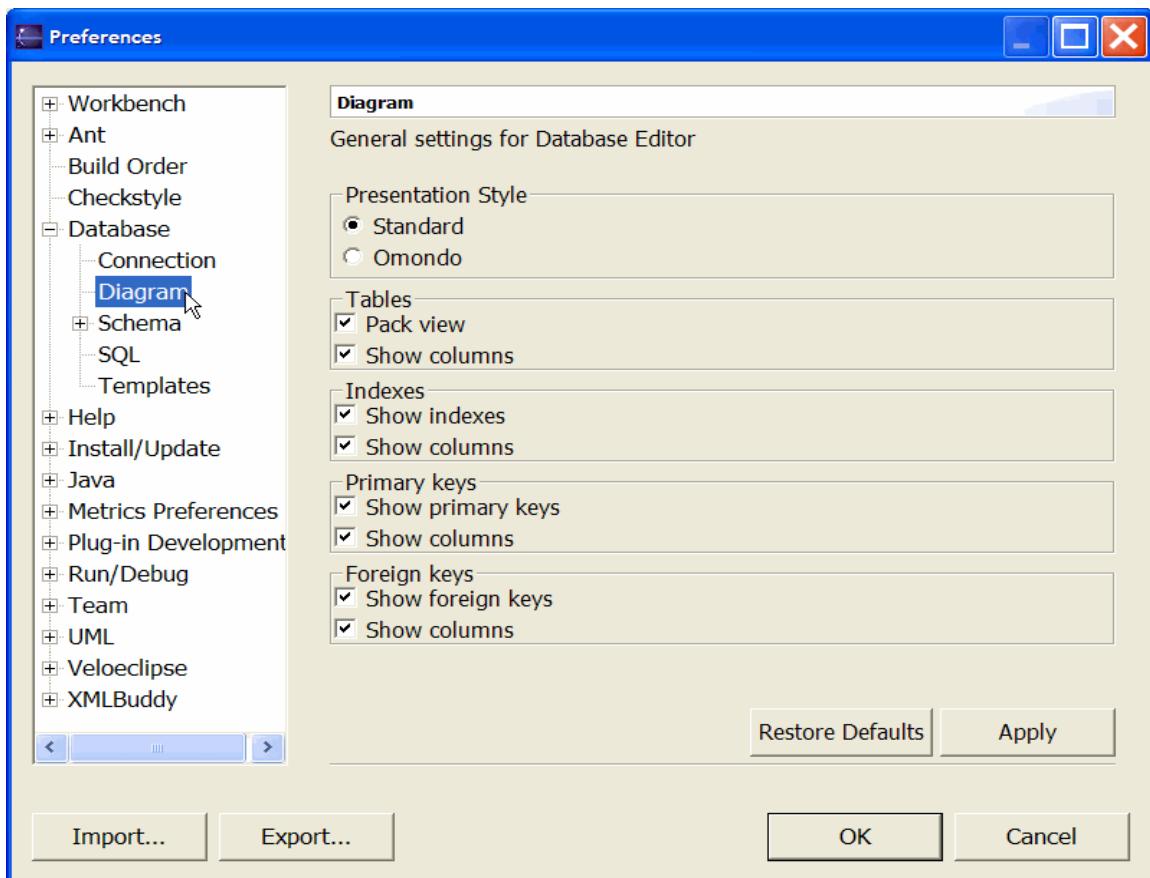
1. Introduction
2. Database Diagram
 1. Presentation Style
 2. Pack View
 3. Show Options

1. Introduction

The Diagram Preferences allows the developer to set Diagram global settings. These settings will be inherited to each new Database Diagram.



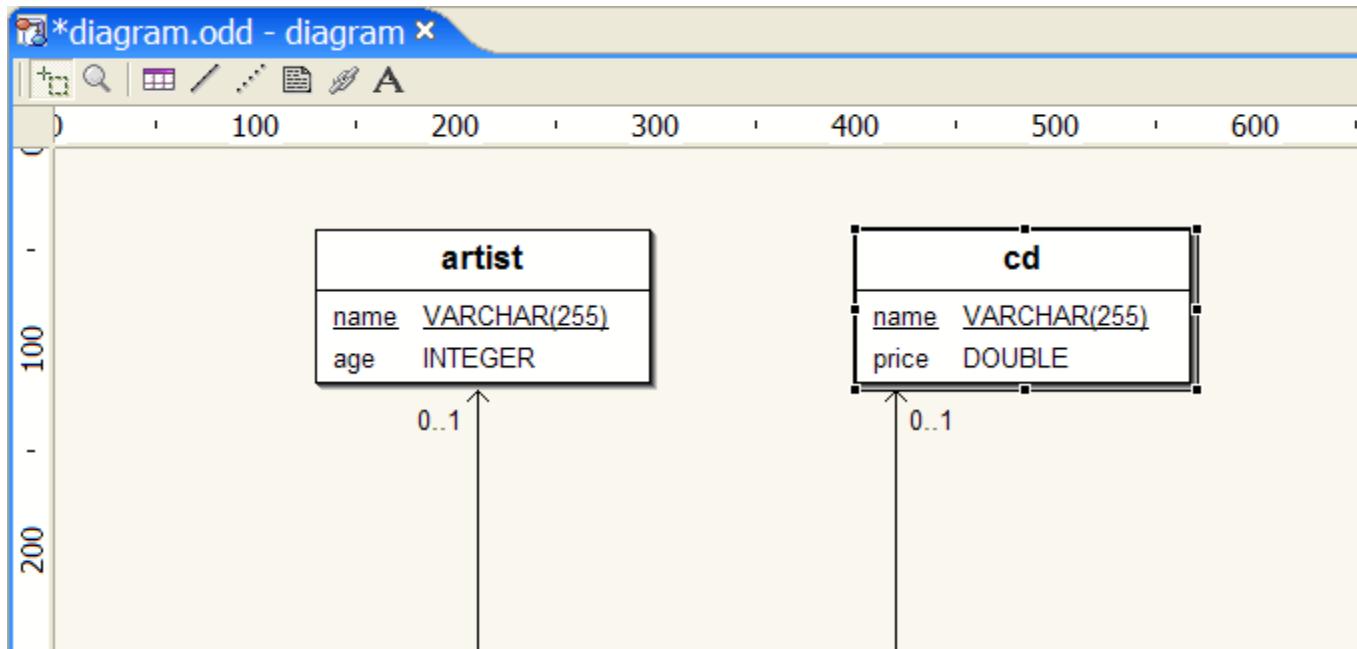
2. Database Diagram



2.1. Presentation Style

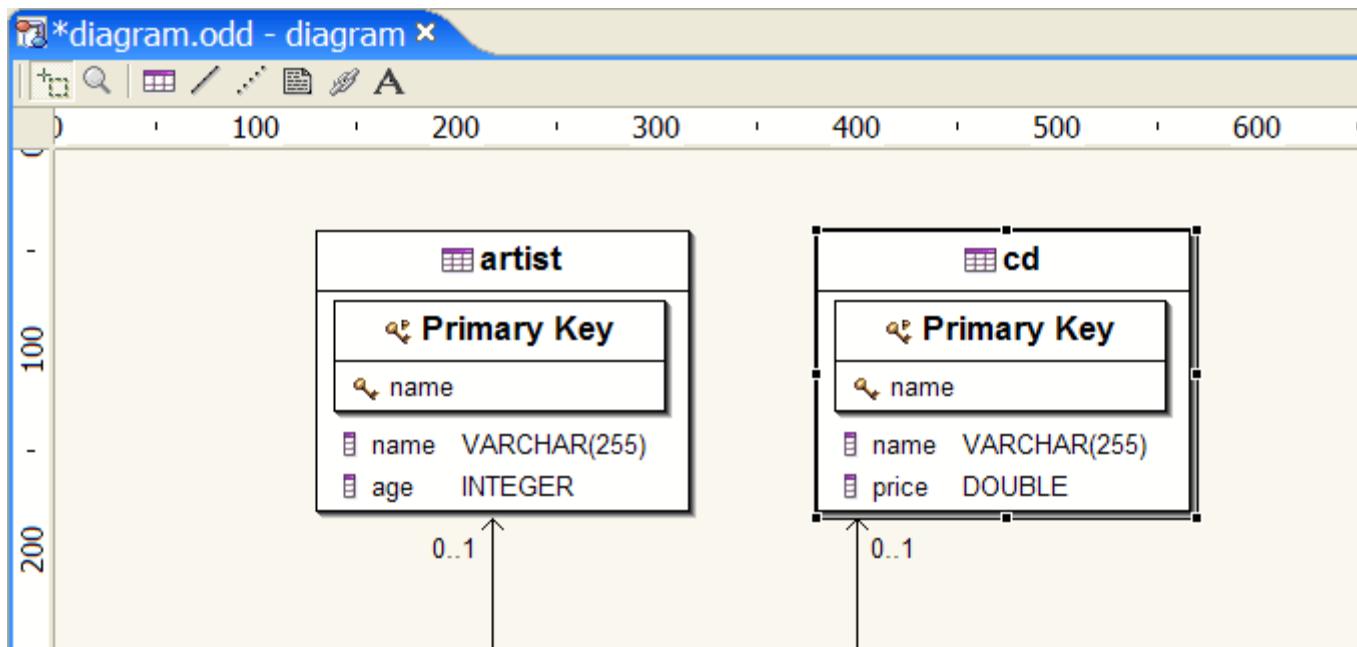
The Presentation Style lets you choose between:

Standard Style :



In the Standard mode, Primary Keys are underlined displayed. Indexes and Foreign Keys are not displayed.

Omondo Style :



2.2. Pack View

The Pack View checked box allows an automatic resize of your table according to its content.

2.3. Show Options

The different Show options will rule the display behaviour in your Diagram.

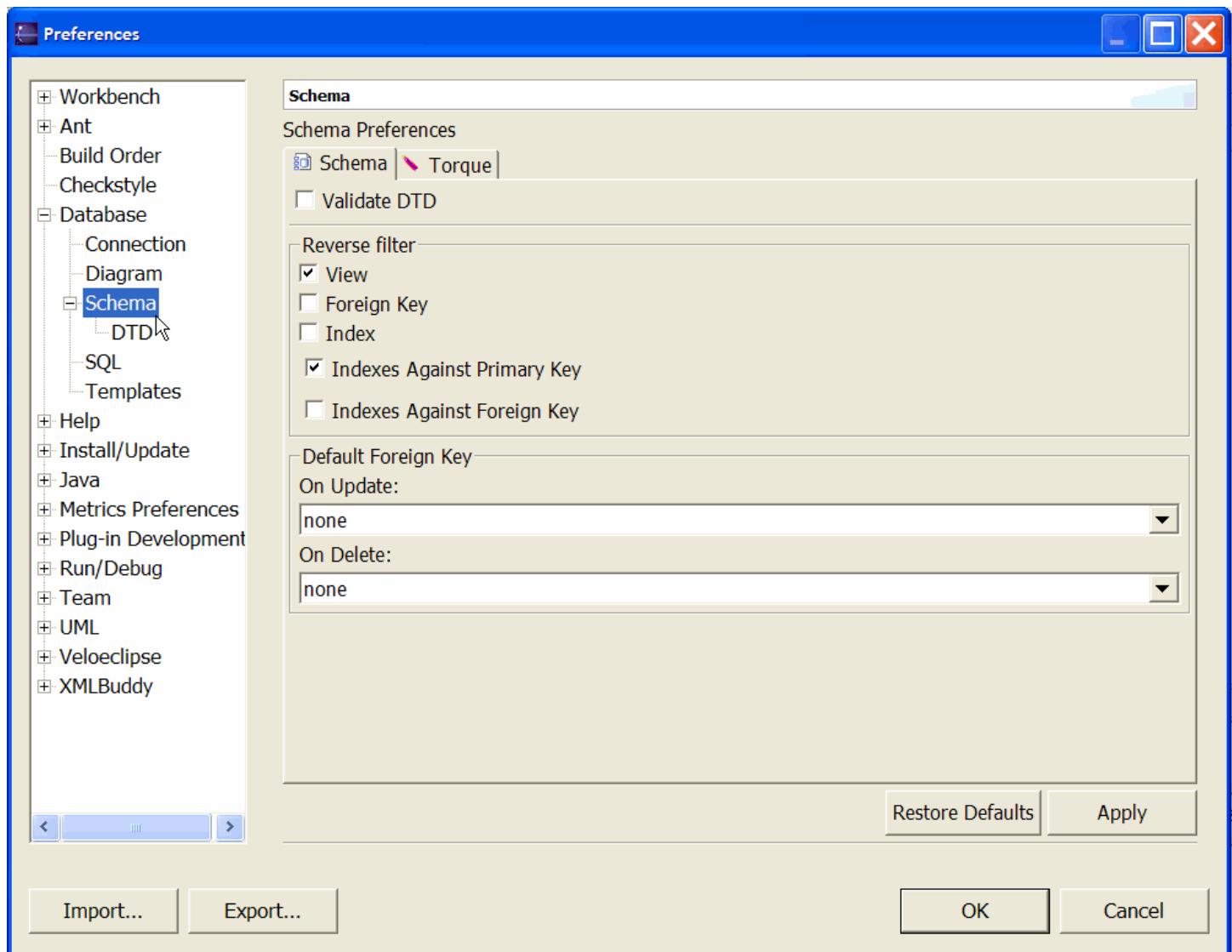
Database Schema

1. Introduction
2. Database Schema
 1. Database Schema DTD
 2. Validate DTD
 3. Reverse Filter
 4. Default Foreign Key
3. Torque

1. Introduction

The Schema Preferences allows the developer to set the Schema global settings. These settings will be inherited to each new Database Schema wizard page.

2. Database Schema



2.2. Validate DTD

Our Database Schema file is associated with a [Database Schema DTD](#).

When this option is activated the DTD will be validated against its associated Schema.

As an example the following screenshot shows the Database Console result when opening an empty schema file.

The associated DTD rules that a database node should have at least one table or view defined.

The screenshot shows three windows in a software interface:

- schema.ods.xml**: Shows XML code for a database node with no tables or views defined.
- Omondo Local DTD.dtd**: Shows the DTD definition for the database element, which requires at least one table or view.
- Database Console**: Shows validation errors. The first message is a status log. The second message is an error: "2004-09-23 15:50:27 [[Error]] schema.ods.xml:4:12: The content of element type "database" is incomplete, it must match "(external-schema*,(table|view)+)"."

2.3. Reverse Filter

The Reverse Filter changes the reverse behaviour when you access the Meta Data of your Database.

When selected :

- Views
- Foreign Key
- Index

are not Reversed

When selected

- Indexes Against Primary Keys
- Indexes Against Foreign Keys

are analysed and filtered.

2.4. Default Foreign Key

You can control the default behaviour of a Foreign Key with this option :

- none
- cascade
- restrict
- setnull

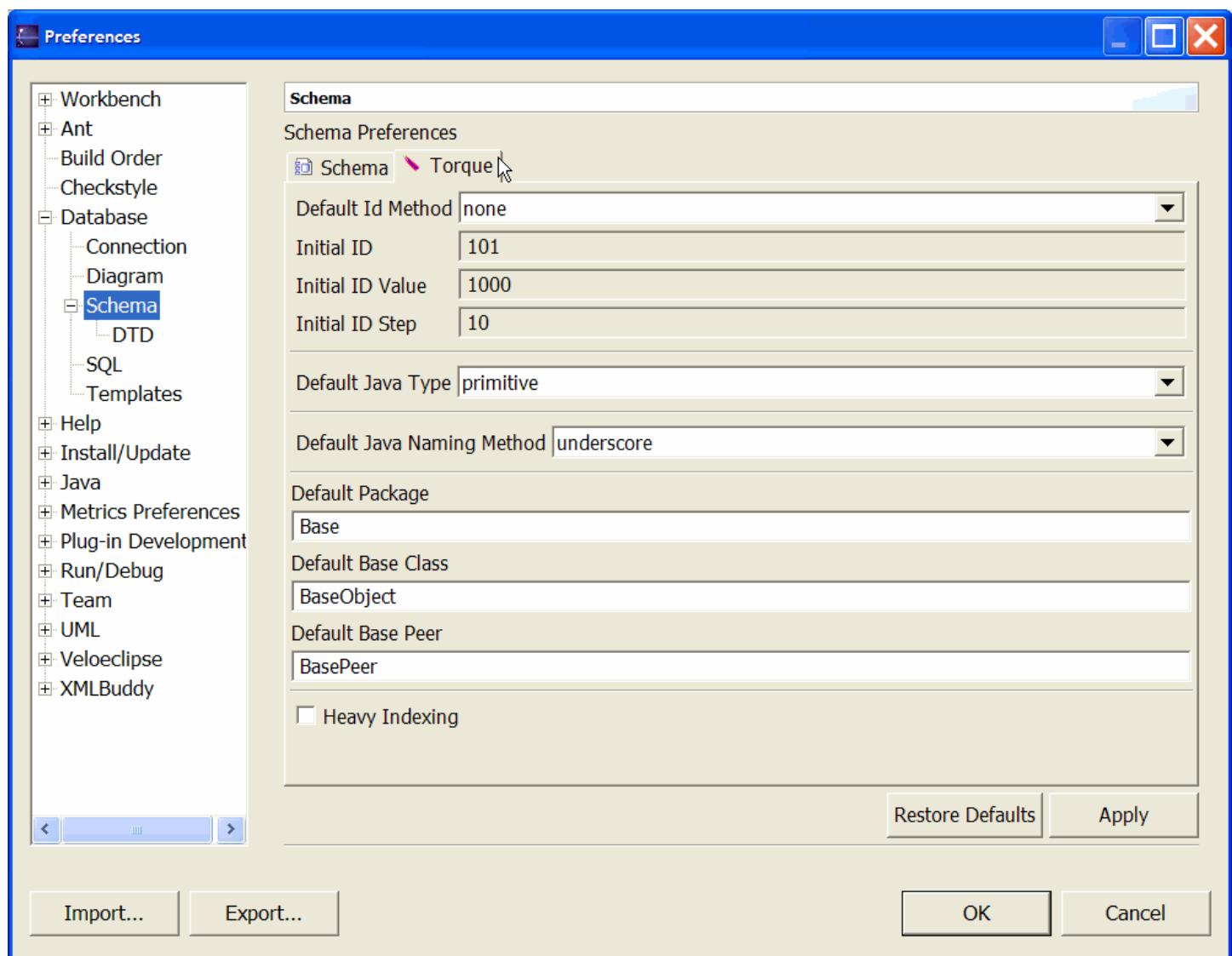
are the available options.

3. Torque

The Database Schema containing all the database physical descriptions is Torque compatible.

If you plan to generate Torque objects, you can set the default Torque options.

For further details about Torque options, you will find a detailed page [here](#).



Database Schema – Database DTD

1. [Introduction](#)
2. [Database Schema DTD](#)

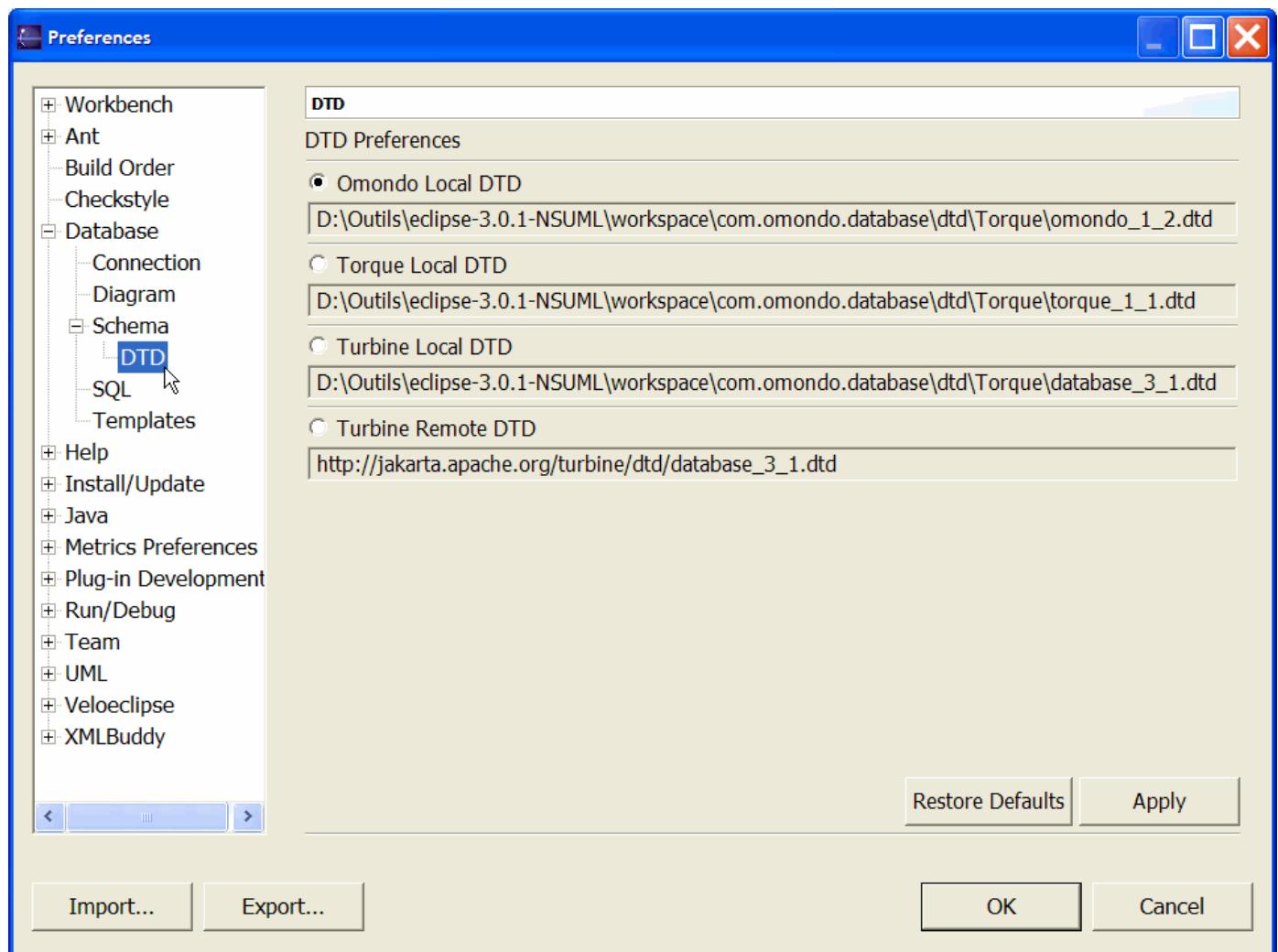
1. Introduction

The DTD Preferences allows the developer to set its DTD global settings. These settings will be inherited to each new Database DTD wizard page.

2. Database Schema DTD

Each Database Schema is associated with its own DTD.

EclipseDatabase allows you to choose among various DTDs to validate the content of your Schemas.



Three DTDs are available :

- the [Omondo DTD](#) is an extended version of the initial Torque DTD.
- the [Torque Local DTD](#) is the current [Apache Torque](#) Project DTD.
- the [Turbine Local DTD](#) is the current [Apache Turbine](#) Project DTD.
- the [Turbine Remote DTD](#) is the remote reference of the previous Turbine Local DTD.

All of this DTDs are compatible with EclipseDatabase.

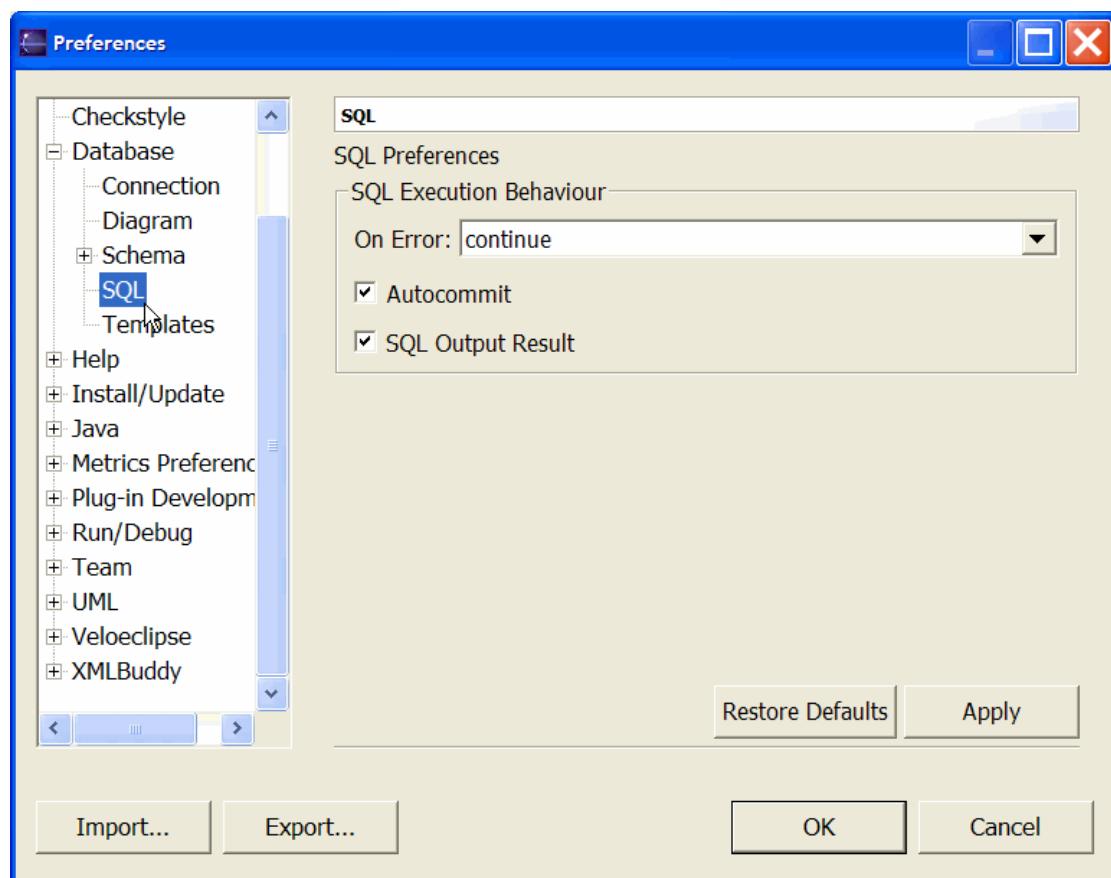
- [DTD Validation](#) can help you check the content of your Schemas.
- XML Editors who have a code completion feature driven by DTD can help your manual edition.

1. [Introduction](#)
2. [Database SQL](#)
 1. [SQL Execution Behaviour](#)
 1. [On Error](#)
 2. [Autocommit](#)
 3. [SQL Output Result](#)

1. Introduction

The SQL Preferences allows the developer to set its SQL execution global settings. These settings will be inherited to each new SQL Forward wizard page.

2. Database SQL



2.1. SQL Execution Behaviour

With the SQL Execution Behaviour set of options you can control the way your SQL instructions are processed.

2.1.1. On Error

With the On Error options you control the flow of your processed SQL instructions.

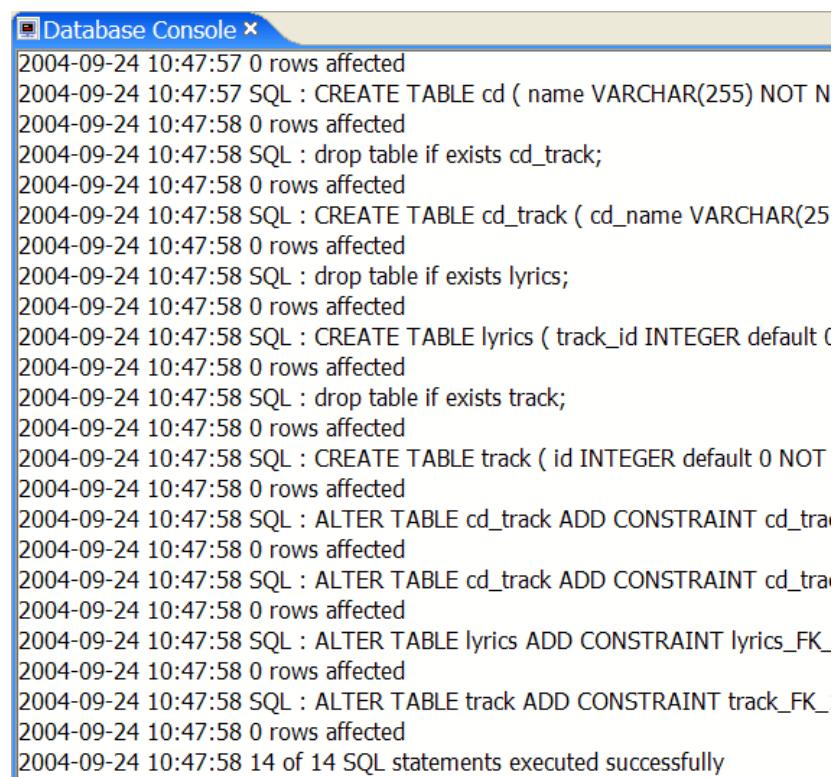
- Continue
- Stop
- Abort

2.1.2. Autocommit

The Autocommit option will consider each SQL instruction as a transaction.

2.1.3. SQL Output Result

The SQL Output Result will display detailed information in the [DatabaseConsole](#) window.



The screenshot shows a window titled "Database Console". The content area displays a log of SQL statements and their execution details:

```

2004-09-24 10:47:57 0 rows affected
2004-09-24 10:47:57 SQL : CREATE TABLE cd ( name VARCHAR(255) NOT N
2004-09-24 10:47:58 0 rows affected
2004-09-24 10:47:58 SQL : drop table if exists cd_track;
2004-09-24 10:47:58 0 rows affected
2004-09-24 10:47:58 SQL : CREATE TABLE cd_track ( cd_name VARCHAR(25
2004-09-24 10:47:58 0 rows affected
2004-09-24 10:47:58 SQL : drop table if exists lyrics;
2004-09-24 10:47:58 0 rows affected
2004-09-24 10:47:58 SQL : CREATE TABLE lyrics ( track_id INTEGER default (
2004-09-24 10:47:58 0 rows affected
2004-09-24 10:47:58 SQL : drop table if exists track;
2004-09-24 10:47:58 0 rows affected
2004-09-24 10:47:58 SQL : CREATE TABLE track ( id INTEGER default 0 NOT
2004-09-24 10:47:58 0 rows affected
2004-09-24 10:47:58 SQL : ALTER TABLE cd_track ADD CONSTRAINT cd_trac
2004-09-24 10:47:58 0 rows affected
2004-09-24 10:47:58 SQL : ALTER TABLE cd_track ADD CONSTRAINT cd_trac
2004-09-24 10:47:58 0 rows affected
2004-09-24 10:47:58 SQL : ALTER TABLE lyrics ADD CONSTRAINT lyrics_FK_
2004-09-24 10:47:58 0 rows affected
2004-09-24 10:47:58 SQL : ALTER TABLE track ADD CONSTRAINT track_FK_
2004-09-24 10:47:58 0 rows affected
2004-09-24 10:47:58 14 of 14 SQL statements executed successfully

```

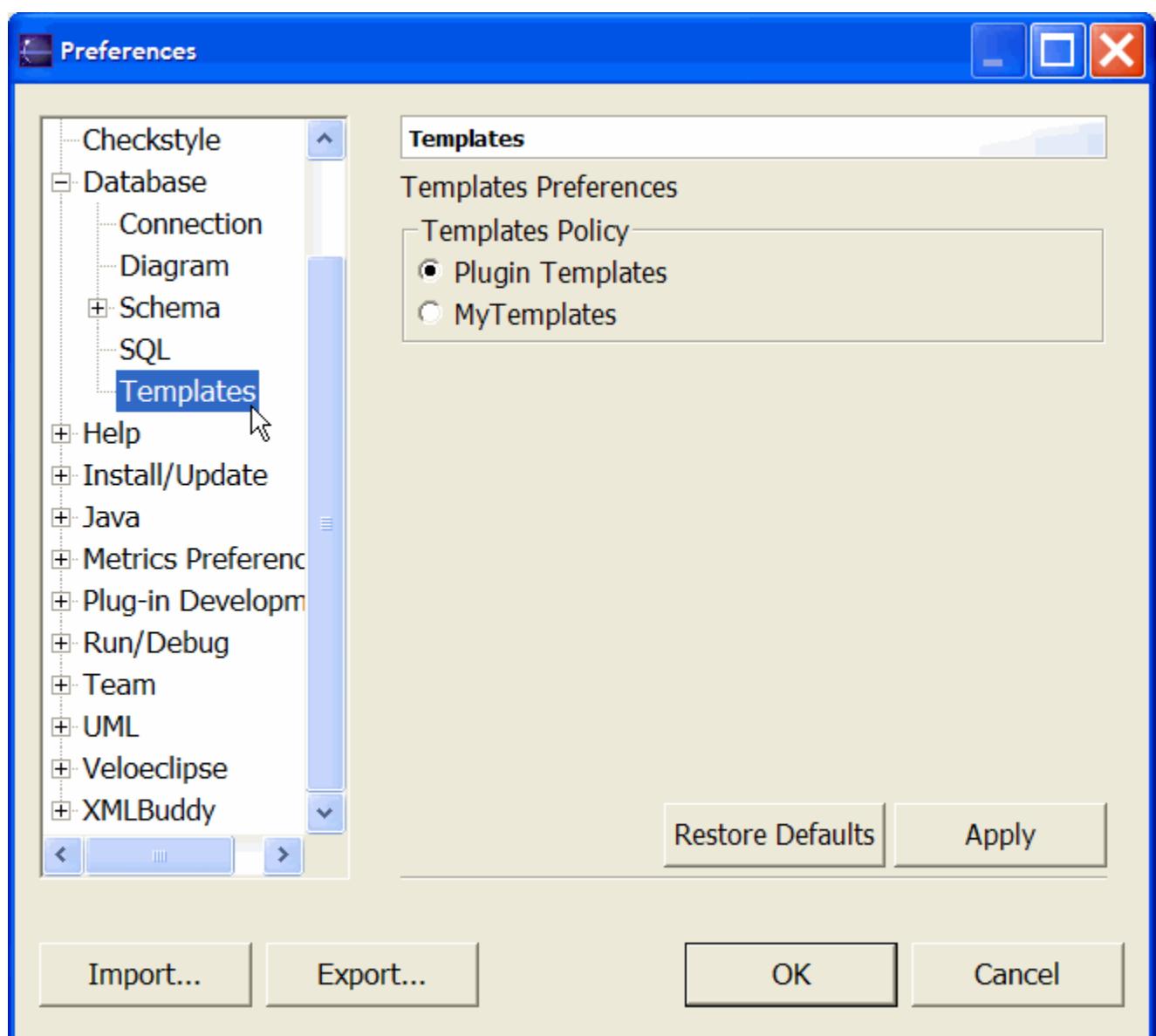
Database Preferences - Templates

1. [Introduction](#)
2. [Templates](#)
 1. [Templates Policy](#)
 1. [Plugin Templates](#)
 2. [MyTemplates](#)

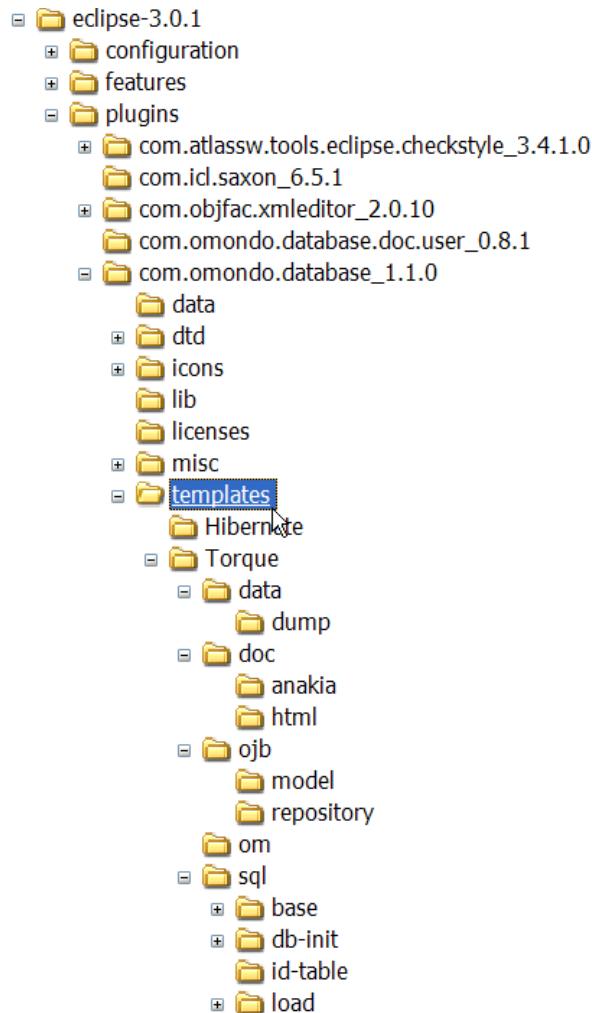
1. Introduction

The Templates Preferences allows the developer to set its Templates global settings. These settings will change the way templates are loaded.

2. Templates



Below your EclipseDatabase plugin installation, the `%ECLIPSE_HOME%/plugins/com.omondo.database_x.y.z/templates` directory contains the plugins templates.



2.1. Templates Policy

One of the benefits of a template mechanism is to allow users to tune or modify them to fit their particular needs.

Without any special feature, users would modify the templates inside the plugin templates directory. This without the benefit of the Eclipse platform :

- Local History
- Team Work
- Compare With

Each time you install a new EclipseDatabase version, these templates could change.

When installing a new version, modified plugin templates would be lost.

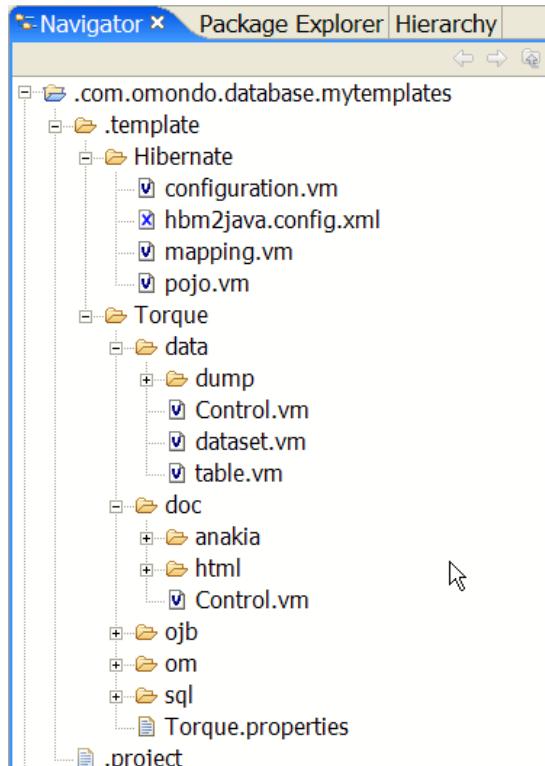
The **MyTemplates** mechanism tries to offer a solution to manipulate the templates with the benefit of the Eclipse platform.

2.1.1. Plugin Templates

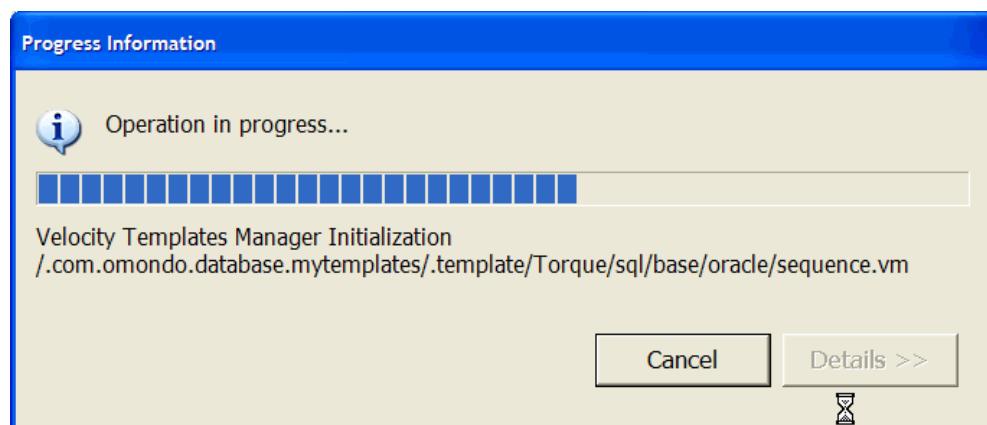
This option lets you use the Plugin templates.

2.1.2. MyTemplates

When this option is selected the Simple project `.com.omondo.database.mytemplates` is created in your workspace.



A copy of your plugin templates installation is made in this project.



Now you can modify your templates with the benefits of your Eclipse Platform.

When you will install a new version of EclipseDatabase a merge mechanism will be processed.

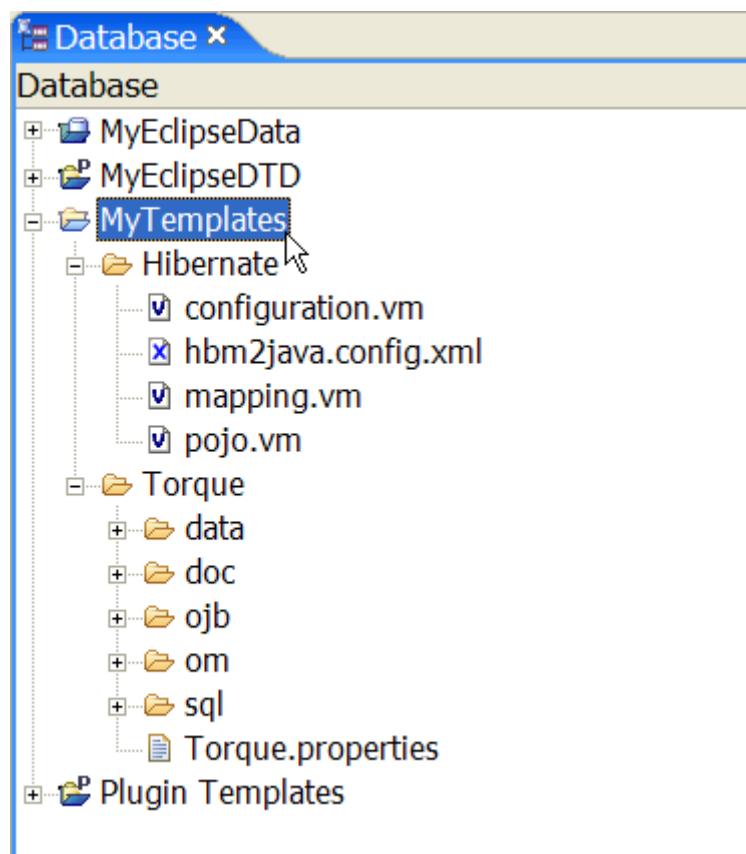
- New templates will be added.
- Old Templates will be kept.
- Updated Templates will remain untouched.

You have noticed that the **.com.omondo.database.mytemplates** starts with a dot.

As a consequence it will not be seen in your Package Explorer.

You need to use the Navigator view to access this project.

A shortcut is available in the [DatabaseExplorer](#).



Database Perspective

1. [Introduction](#)
2. [Database Perspective](#)
3. [Database Explorer](#)
4. [Database Console](#)

1. Introduction

The Database Perspective allows the developer to focus only on the database tasks.

The major advantage of using Database perspective is to organize a predefined set of database specialized screens

- Database Explorer
- Database Console
- Database Properties
- Package Explorer
- Navigator
- Editors
- Outline
- Tasks

Each major Eclipse perspective has some predefined Database shortcuts

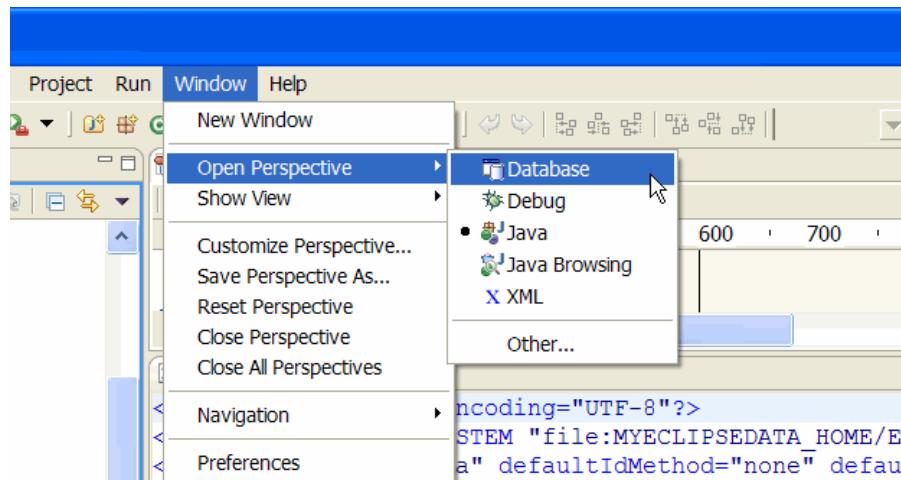
- Resource
- Java
- Debug
- Java Browsing
- Java Type Hierarchy
- CVS Repository Exploring
- Plug-in Development

to:

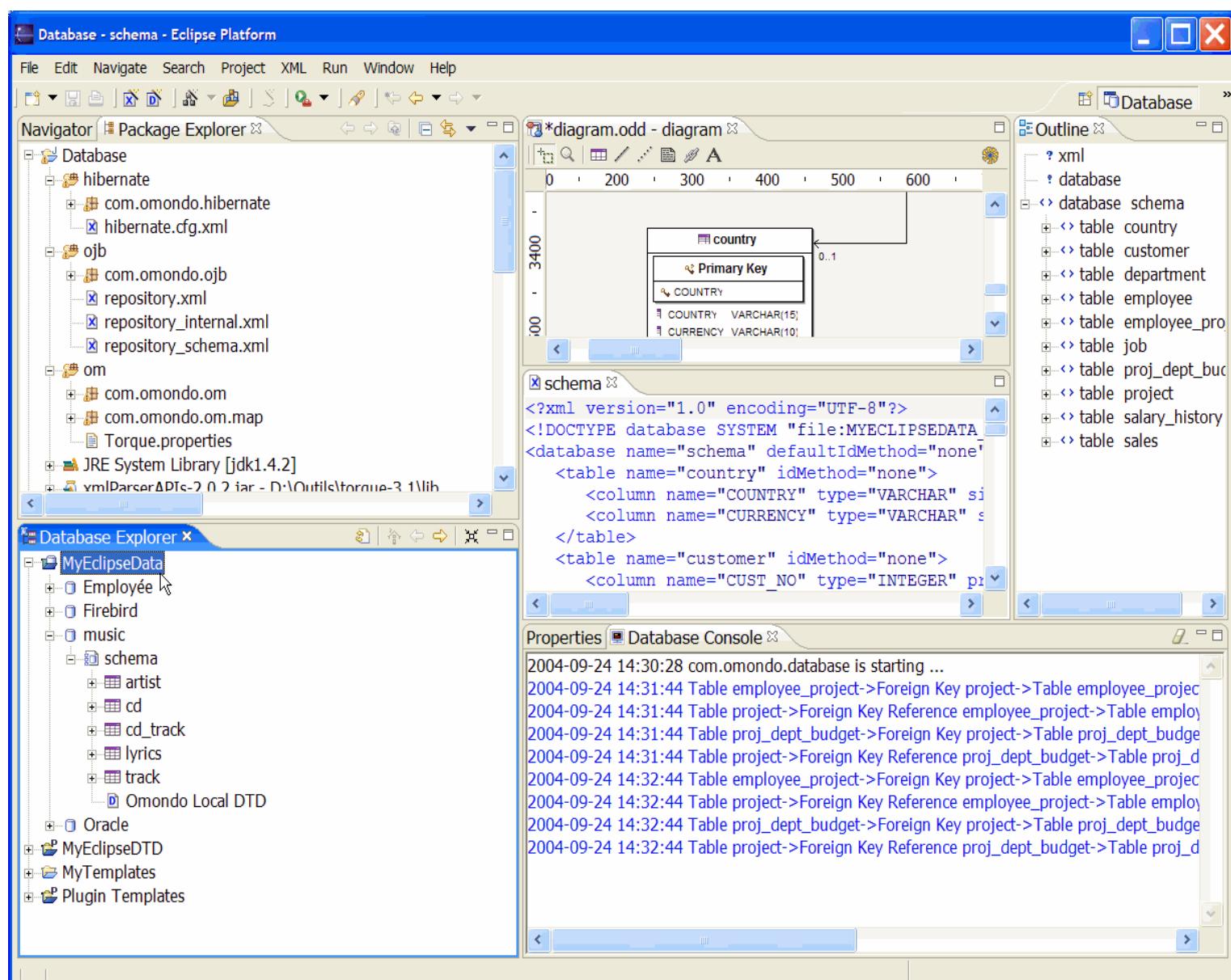
- **Window -> Open Perspective -> Database**
- **Window -> Show View -> Database Explorer**
- **Window -> Show View -> Database Console**

2. Database Perspective

In the window menu select Window -> Open Perspective -> Database.



Open the Database Perspective.



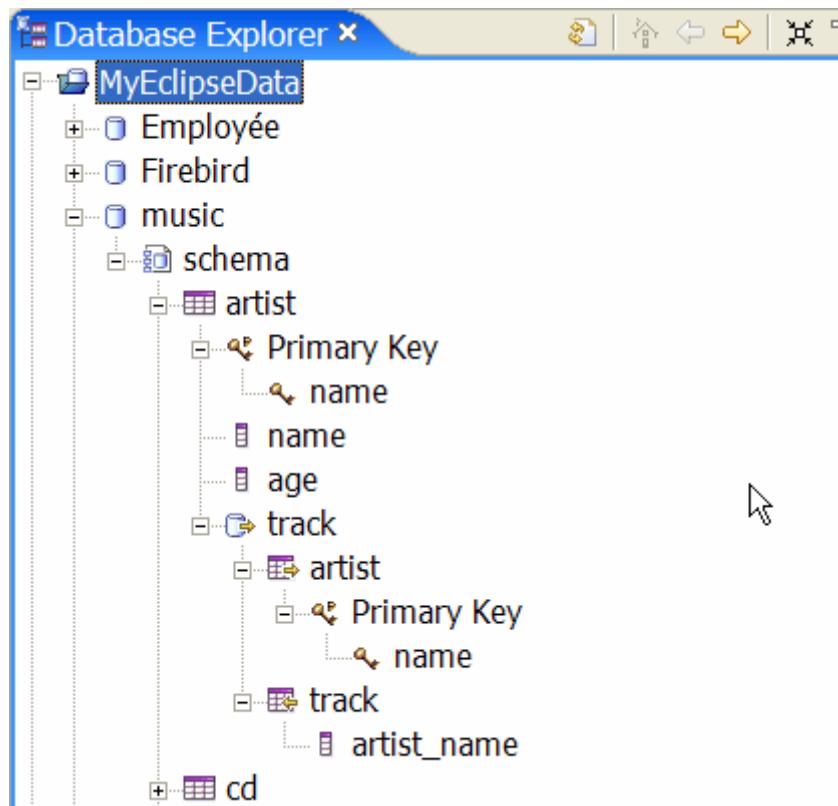
DatabaseExplorer

1. [Introduction](#)
2. [MyEclipseData](#)
3. [MyEclipseDTD](#)
4. [MyTemplates](#)
5. [Plugin Templates](#)

1. Introduction

The DatabaseExplorer is a view which gathers static and dynamic database informations.

2. MyEclipseData



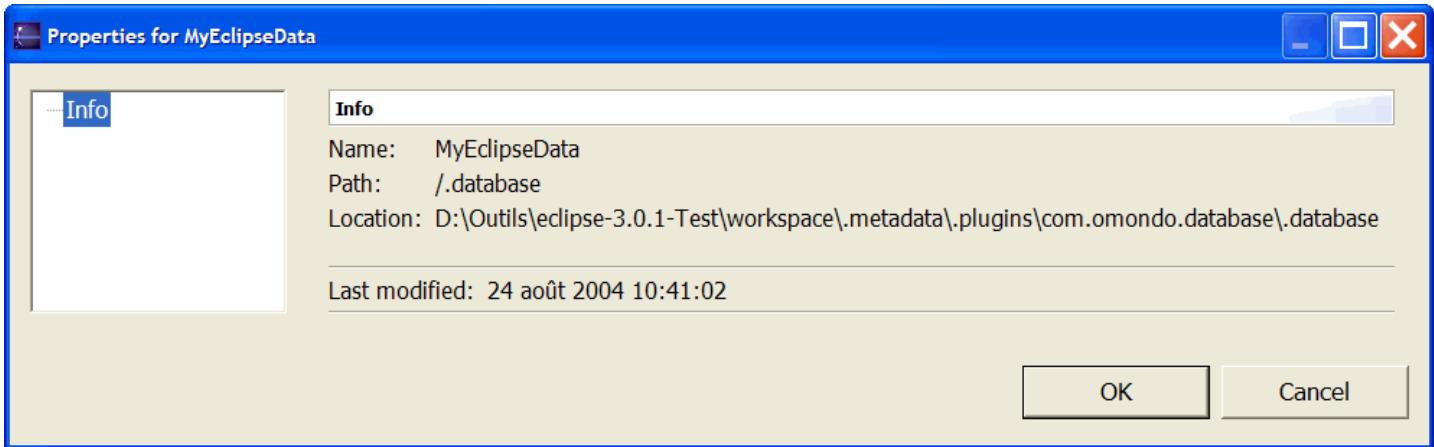
MyEclipseData is a view of your existing databases.

The Connection you can create will attempt to retrieve the Database Meta Data from an existing Database.

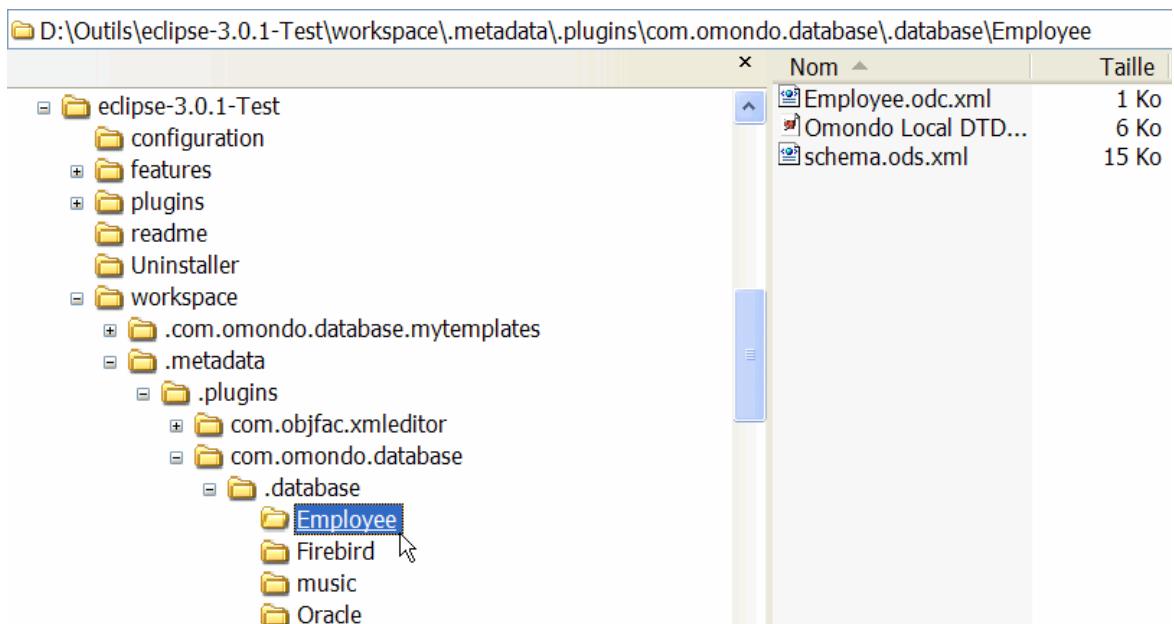
You cannot design a Database Model from MyEclipseData.

You need to design your Database Model from a Java Project or a Simple Project in the workspace. The MyEclipseData information doesn't belong to the workspace but is stored in the EclipseDatabase plugin metadata area :

%WORKSPACE_HOME%/.metadata/.plugins/com.omondo.database



In this area you will find all the stored information.

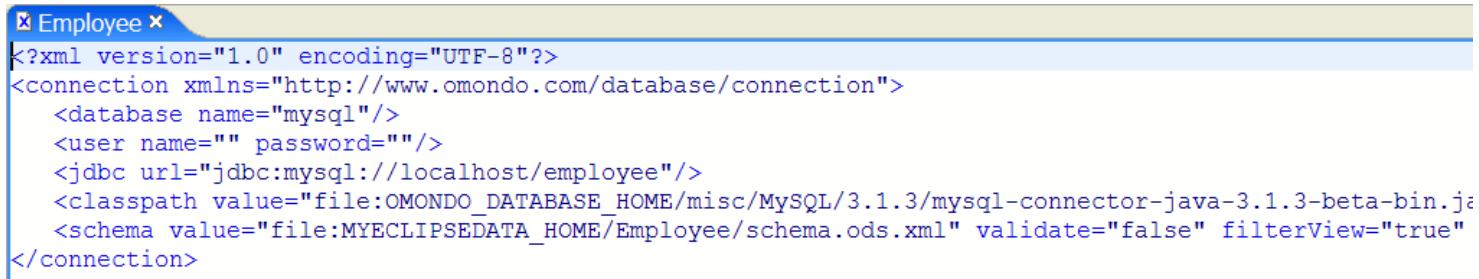


Beware that even though the **.metadata** directory is below the **%WORKSPACE_HOME%** directory its content doesn't belong to the workspace.

The content of this directory is not browsable from your Package Explorer or your Navigator view.

If you want to keep these files for further processing, the structure of this **.database** directory is platform independent.

The Connection directory and the Connection should have the same name.



```

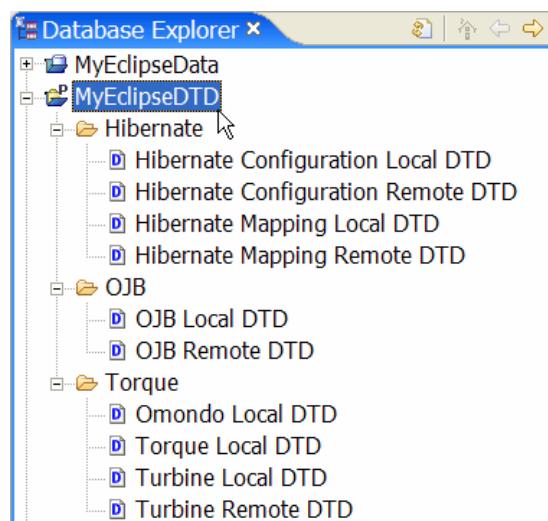
<?xml version="1.0" encoding="UTF-8"?>
<connection xmlns="http://www.omondo.com/database/connection">
  <database name="mysql"/>
  <user name="" password="" />
  <jdbc url="jdbc:mysql://localhost/employee"/>
  <classpath value="file:OMONDO_DATABASE_HOME/misc/MySQL/3.1.3/mysql-connector-java-3.1.3-beta-bin.jar" />
  <schema value="file:MYECLIPSEDATA_HOME/Employee/schema.ods.xml" validate="false" filterView="true" />
</connection>

```

The [OMONDO DATABASE HOME](#) is an internal variable.

The [MYECLIPSEDATA_HOME](#) is an internal variable which is a shortcut to the MyEclipseData location.

3. MyEclipseDTD



The MyEclipseDTD area is a shortcut to the known EclipseDatabase DTDs.

Most of the DTDs are locally or remotely accessible.

Hibernate

- o Hibernate Configuration
- o Hibernate Mapping

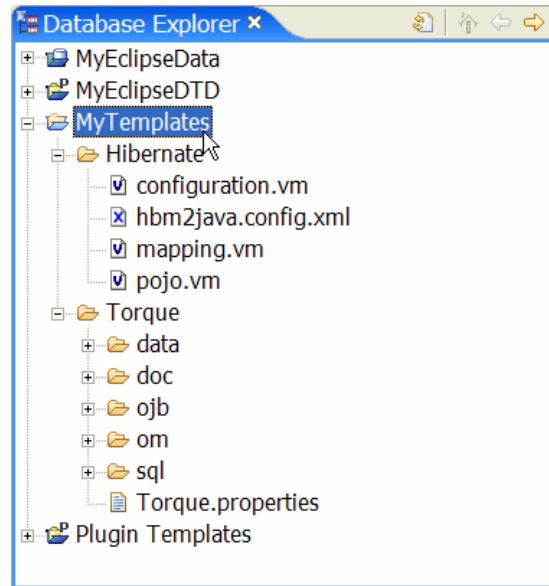
Object Relational Bridge

- o OJB

[Schema and DTDs](#)

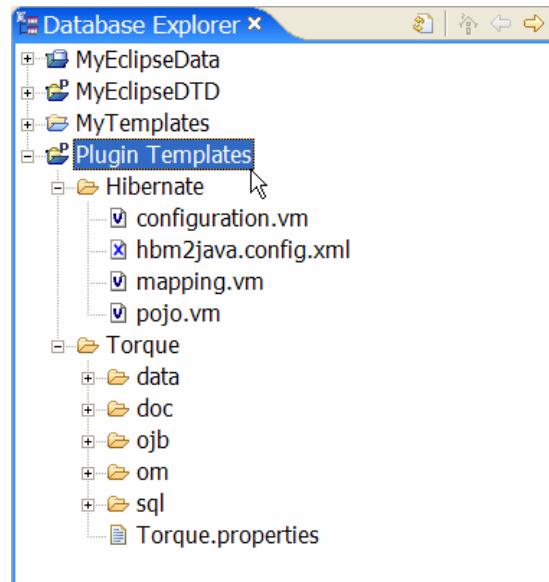
- o Omondo
- o Torque
- o Turbine

4. MyTemplates



The [MyTemplates](#) area is a shortcut to the `.com.omondo.database.mytemplates` Simple Project.

5. Plugin Templates



The [Plugin Templates](#) area is a shortcut to the EclipseDatabase plugin templates.

1. Introduction
2. DatabaseConsole
 1. Rubber
 2. Cut and Paste
 3. Eclipse Log File

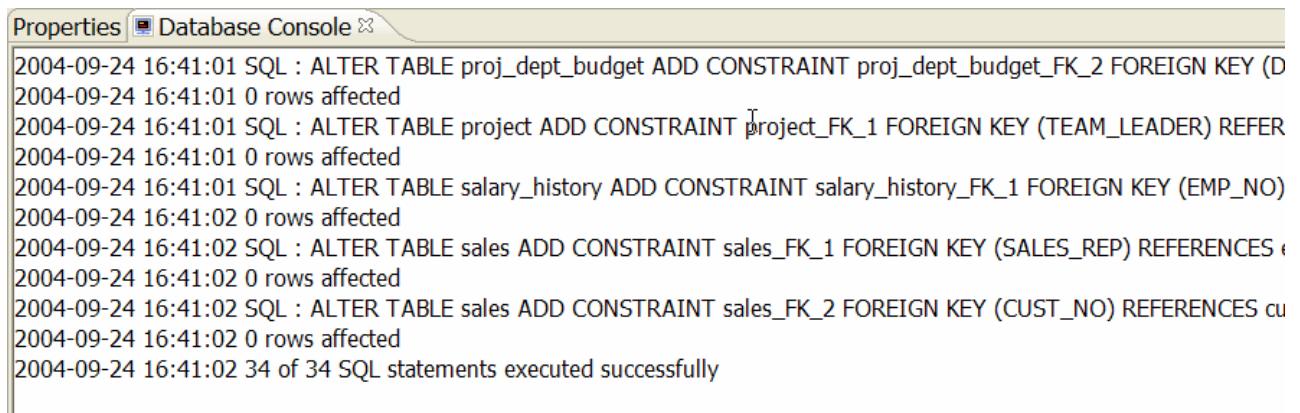
1. Introduction

The Database Console is used to have live information on what is happening.

2. DatabaseConsole

Different kinds of output can be reported in the DatabaseConsole.

- SQL Output



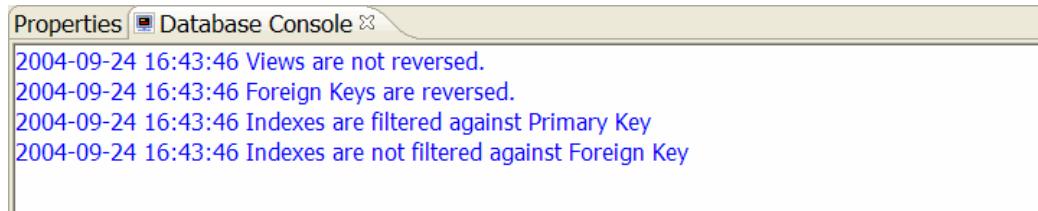
Properties Database Console

```

2004-09-24 16:41:01 SQL : ALTER TABLE proj_dept_budget ADD CONSTRAINT proj_dept_budget_FK_2 FOREIGN KEY (D
2004-09-24 16:41:01 0 rows affected
2004-09-24 16:41:01 SQL : ALTER TABLE project ADD CONSTRAINT project_FK_1 FOREIGN KEY (TEAM_LEADER) REFER
2004-09-24 16:41:01 0 rows affected
2004-09-24 16:41:01 SQL : ALTER TABLE salary_history ADD CONSTRAINT salary_history_FK_1 FOREIGN KEY (EMP_NO)
2004-09-24 16:41:02 0 rows affected
2004-09-24 16:41:02 SQL : ALTER TABLE sales ADD CONSTRAINT sales_FK_1 FOREIGN KEY (SALES REP) REFERENCES (
2004-09-24 16:41:02 0 rows affected
2004-09-24 16:41:02 SQL : ALTER TABLE sales ADD CONSTRAINT sales_FK_2 FOREIGN KEY (CUST_NO) REFERENCES cu
2004-09-24 16:41:02 0 rows affected
2004-09-24 16:41:02 34 of 34 SQL statements executed successfully

```

- Reverse Output



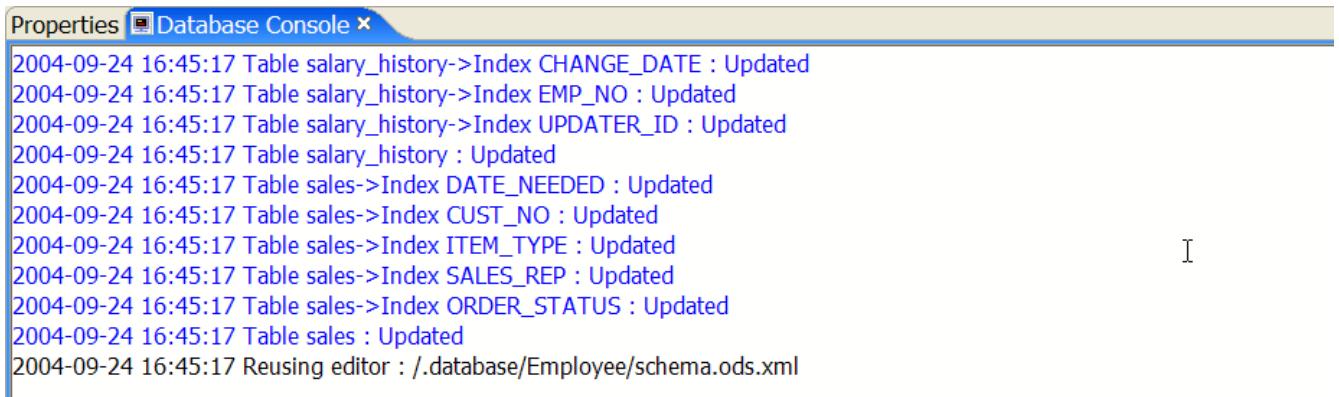
Properties Database Console

```

2004-09-24 16:43:46 Views are not reversed.
2004-09-24 16:43:46 Foreign Keys are reversed.
2004-09-24 16:43:46 Indexes are filtered against Primary Key
2004-09-24 16:43:46 Indexes are not filtered against Foreign Key

```

- Merged Reverse Output



Properties Database Console

```

2004-09-24 16:45:17 Table salary_history->Index CHANGE_DATE : Updated
2004-09-24 16:45:17 Table salary_history->Index EMP_NO : Updated
2004-09-24 16:45:17 Table salary_history->Index UPDATER_ID : Updated
2004-09-24 16:45:17 Table salary_history : Updated
2004-09-24 16:45:17 Table sales->Index DATE_NEEDED : Updated
2004-09-24 16:45:17 Table sales->Index CUST_NO : Updated
2004-09-24 16:45:17 Table sales->Index ITEM_TYPE : Updated
2004-09-24 16:45:17 Table sales->Index SALES REP : Updated
2004-09-24 16:45:17 Table sales->Index ORDER_STATUS : Updated
2004-09-24 16:45:17 Table sales : Updated
2004-09-24 16:45:17 Reusing editor : ./database/Employee/schema.ods.xml

```

- Templates Generation Report

```

Properties Database Console
2004-09-24 16:47:38 Error in evaluation of == expression. Both arguments must be of the same Class. Currently left = class java.lang.Object
2004-09-24 16:47:38 RHS of #set statement is null. Context will not be modified. om/Object.vm [line 268, column 1]
2004-09-24 16:47:38 Error in evaluation of == expression. Both arguments must be of the same Class. Currently left = class java.lang.Object
2004-09-24 16:47:38 Error in evaluation of == expression. Both arguments must be of the same Class. Currently left = class java.lang.Object
2004-09-24 16:47:38 Error in evaluation of == expression. Both arguments must be of the same Class. Currently left = class java.lang.Object
2004-09-24 16:47:38 Error in evaluation of == expression. Both arguments must be of the same Class. Currently left = class java.lang.Object
2004-09-24 16:47:38 RHS of #set statement is null. Context will not be modified. om/Object.vm [line 268, column 1]
2004-09-24 16:47:38 Error in evaluation of == expression. Both arguments must be of the same Class. Currently left = class java.lang.Object
2004-09-24 16:47:38 Error in evaluation of == expression. Both arguments must be of the same Class. Currently left = class java.lang.Object
2004-09-24 16:47:38 RHS of #set statement is null. Context will not be modified. om/Object.vm [line 268, column 1]

```

2.1. Rubber

You can use the rubber to clear the console.

2.2. Cut and Paste

```

Properties Database Console
2004-09-24 16:51:17 Views are not reversed.
2004-09-24 16:51:17 Foreign Keys are reversed.
2004-09-24 16:51:17 Indexes are filtered against Primary Key
2004-09-24 16:51:17 Table employee_project->Foreign Key project->Table employee_project->Column PROJ_ID->Table project->Primary Key->Column PR
2004-09-24 16:51:17 Table project->Foreign Key Reference employee_project->Table employee_project->Column PROJ_ID->Table project->Primary Key->Column PR
2004-09-24 16:51:17 Table proj_dept_budget->Foreign Key project->Table proj_dept_budget->Column PROJ_ID->Table project->Primary Key->Column PR
2004-09-24 16:51:17 Table project->Foreign Key Reference proj_dept_budget->Table proj_dept_budget->Column PROJ_ID->Table project->Primary Key->Column PR
2004-09-24 16:51:17 Indexes are not filtered against Foreign Key
2004-09-24 16:51:17 Reusing editor : ./database/Employee/schema.ods.xml

1*
Project View Format Column Macro Advanced Window Help

```

The DatabaseConsole content can be Cut and Pasted to other tools.

2.3. Eclipse Log File

```

!MESSAGE The content type with id "com.objfac.xmlbuddy.xmlfile" specified in the extension point does not exist.

!ENTRY com.omondo.database 4 0 sept. 24, 2004 17:00:37.707
!MESSAGE created new instance of class org.apache.xerces.parsers.IntegratedParserConfiguration using ClassLoader: org.eclipse.

!ENTRY com.omondo.database 4 0 sept. 24, 2004 17:00:37.707
!MESSAGE Database Connection file loaded D:\Outils\eclipse-3.0.1-Test\workspace\.metadata\.plugins\com.omondo.database\.database

!ENTRY com.omondo.database 4 0 sept. 24, 2004 17:00:37.707
!MESSAGE created new instance of class org.apache.xerces.parsers.IntegratedParserConfiguration using ClassLoader: org.eclipse.

!ENTRY com.omondo.database 4 0 sept. 24, 2004 17:00:37.723
!MESSAGE Database Connection file loaded D:\Outils\eclipse-3.0.1-Test\workspace\.metadata\.plugins\com.omondo.database\.database

!ENTRY com.omondo.database 4 0 sept. 24, 2004 17:00:37.723
!MESSAGE created new instance of class org.apache.xerces.parsers.IntegratedParserConfiguration using ClassLoader: org.eclipse.

!ENTRY com.omondo.database 4 0 sept. 24, 2004 17:00:37.723
!MESSAGE Database Connection file loaded D:\Outils\eclipse-3.0.1-Test\workspace\.metadata\.plugins\com.omondo.database\.database

!ENTRY com.omondo.database 4 0 sept. 24, 2004 17:00:37.739
!MESSAGE created new instance of class org.apache.xerces.parsers.IntegratedParserConfiguration using ClassLoader: org.eclipse.

!ENTRY com.omondo.database 4 0 sept. 24, 2004 17:00:37.739
!MESSAGE Database Connection file loaded D:\Outils\eclipse-3.0.1-Test\workspace\.metadata\.plugins\com.omondo.database\.database

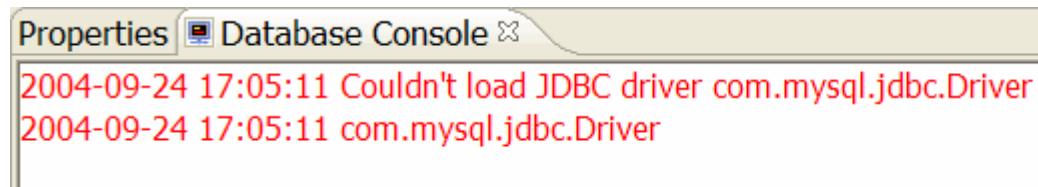
!ENTRY com.omondo.database 4 0 sept. 24, 2004 17:00:37.739
!MESSAGE created new instance of class org.apache.xerces.parsers.IntegratedParserConfiguration using ClassLoader: org.eclipse.

!ENTRY com.omondo.database 4 0 sept. 24, 2004 17:00:37.739
!MESSAGE Database Connection file loaded D:\Outils\eclipse-3.0.1-Test\workspace\.metadata\.plugins\com.omondo.database\.database

```

When started in debug mode, the DatabaseConsole output is also dumped in the **%ECLIPSE_HOME%/workspace/.metadata/.log** file.

However errors are always dumped either in the DatabaseConsole :



or in the Eclipse log file.

```

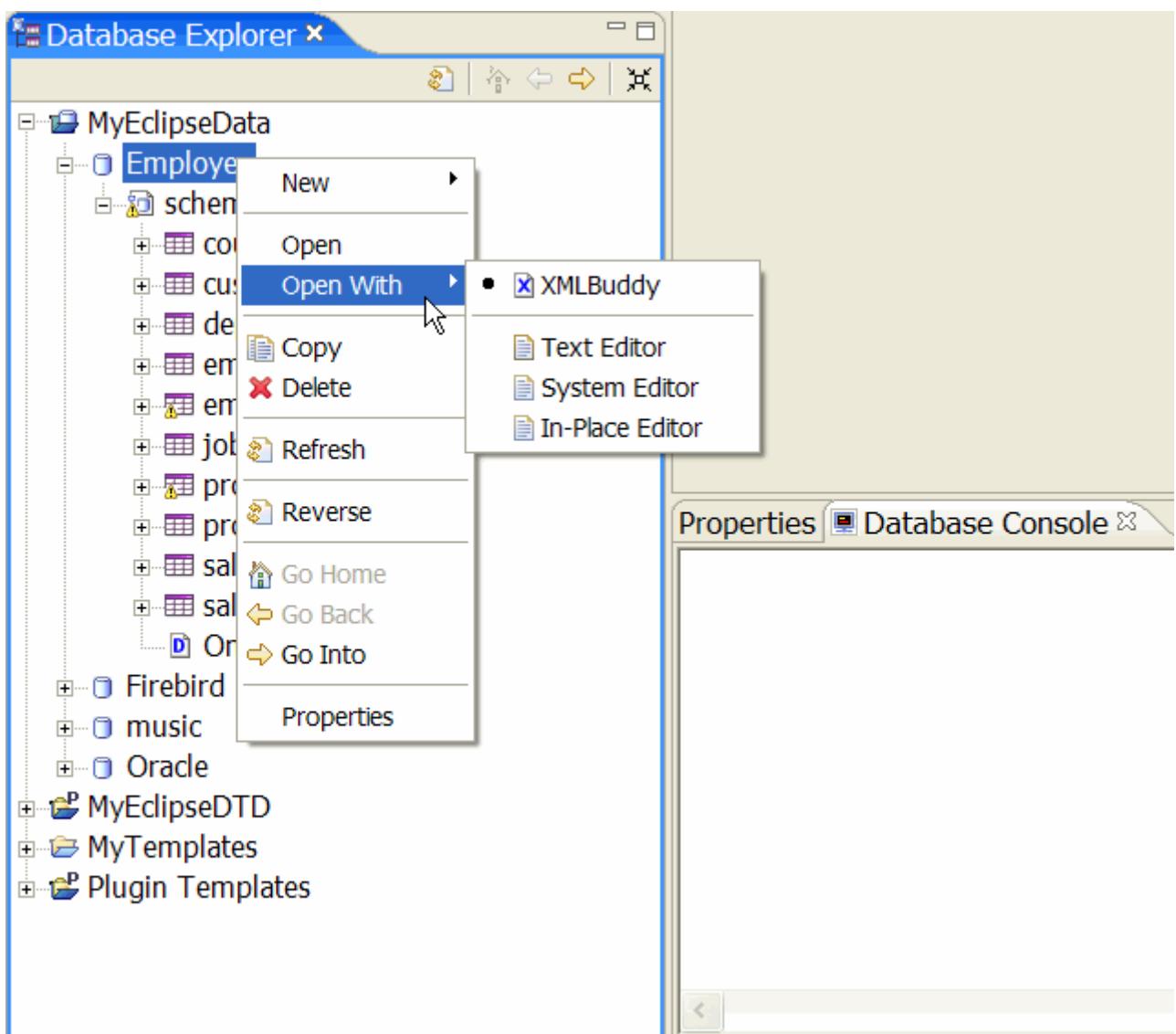
!ENTRY com.omondo.database 4 0 sept. 24, 2004 17:05:11.442
!MESSAGE Couldn't load JDBC driver com.mysql.jdbc.Driver [com.mysql.jdbc.Driver]
!STACK 0
java.lang.ClassNotFoundException: com.mysql.jdbc.Driver
    at java.net.URLClassLoader$1.run(URLClassLoader.java:199)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:187)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:289)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:235)
    at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:302)
    at java.lang.Class.forName0(Native Method)
    at java.lang.Class.forName(Class.java:219)
    at com.omondo.database.b.l.a(SourceFile:192)
    at com.omondo.database.c.a.a(SourceFile:81)
    at com.omondo.database.h.h.bp.do(SourceFile:498)
    at com.omondo.database.h.h.al.int(SourceFile:348)
    at com.omondo.database.h.h.f.widgetSelected(SourceFile:274)
    at org.eclipse.swt.widgets.TypedListener.handleEvent(TypedListener.java:89)
    at org.eclipse.swt.widgets.EventTable.sendEvent(EventTable.java:82)
    at org.eclipse.swt.widgets.Widget.sendEvent(Widget.java:796)
    at org.eclipse.swt.widgets.Display.runDeferredEvents(Display.java:2772)
    at org.eclipse.swt.widgets.Display.readAndDispatch(Display.java:2431)
    at org.eclipse.jface.window.Window.runEventLoop(Window.java:668)
    at org.eclipse.jface.window.Window.open(Window.java:648)
    at org.eclipse.ui.dialogs.PropertyDialogAction.run(PropertyDialogAction.java:177)
    at org.eclipse.jface.action.Action.runWithEvent(Action.java:881)
    at org.eclipse.jface.action.ActionContributionItem.handleWidgetSelection(ActionContributionItem.java:915)
    at org.eclipse.jface.action.ActionContributionItem.access$2(ActionContributionItem.java:866)
    at org.eclipse.jface.action.ActionContributionItem$7.handleEvent(ActionContributionItem.java:785)
    at org.eclipse.swt.widgets.EventTable.sendEvent(EventTable.java:82)
    at org.eclipse.swt.widgets.Widget.sendEvent(Widget.java:796)

```

Working with Database Explorer

1. Introduction
2. General Commands
 1. Open and Open With
 2. Refresh
 3. Delete
 4. Collapse All and Expand All
 5. Home, Go Home, Back, Go Back and Go Into
 6. Properties
3. Properties View
4. Database Connection

1. Introduction

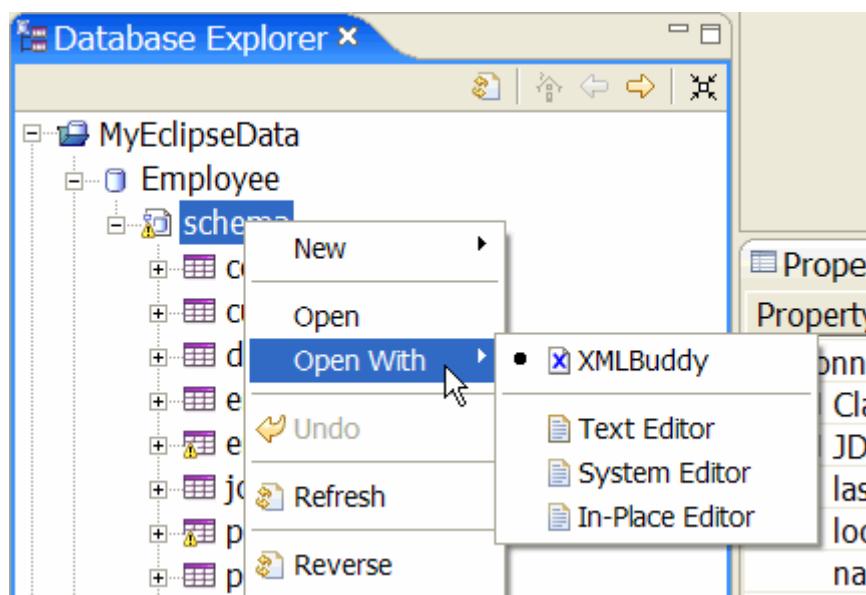


2. General Commands

The Database Explorer contains a set of specialized commands.

- Open and Open With
- Refresh
- Delete
- Collapse All and Expand All
- Home, Go Home, Back, Go Back and Go Into
- Properties

2.1. Open and Open With



The **Open** and **Open With** command have the same behaviour as the workspace commands.

The **Open** command opens the associated default editor.

The **Open With** command lets you choose among the associated editors :

Standard file extensions

- .sql
- .dtd
- .xml
- .properties

or EclipseDatabase file extensions

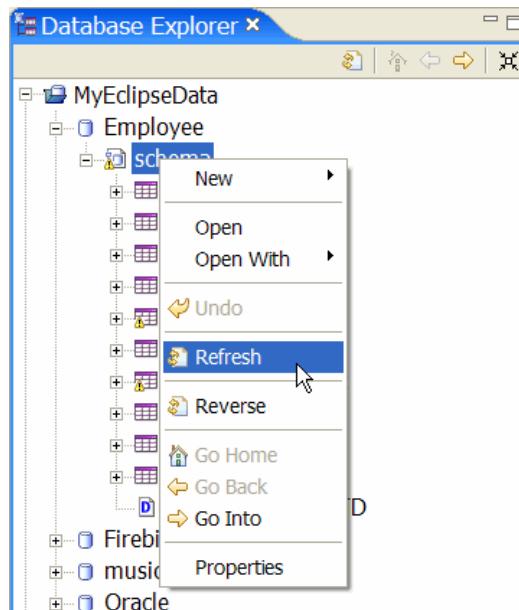
- .odc
- .ods
- .odd

EclipseDatabase file extensions can be managed either with an xml or with their defined extension through a **Save Policy** Global Preference.

Files opened through these commands are :

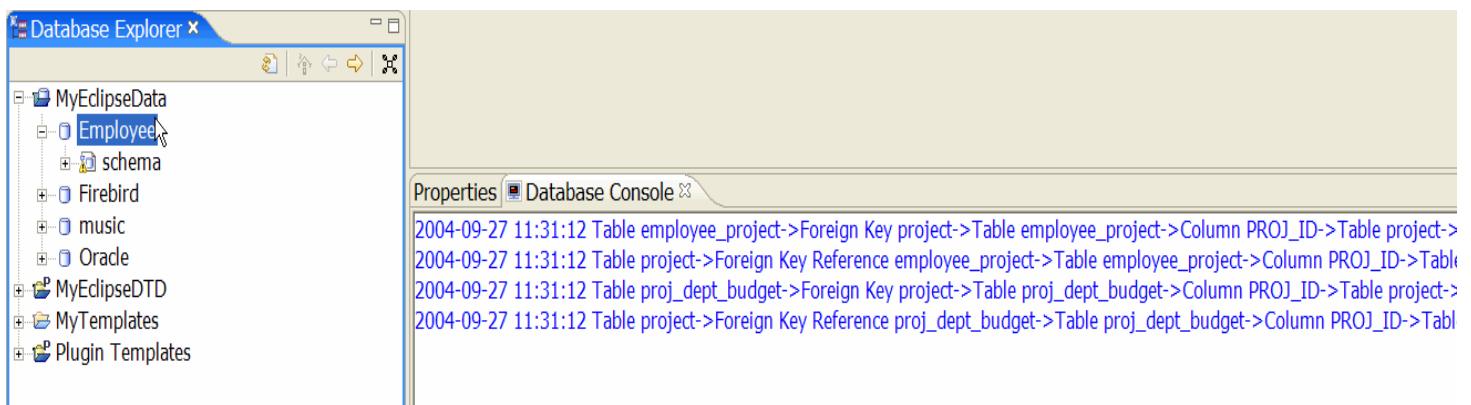
- **MyEclipseData**, in Read only mode.
- **MyEclipseDTD**, in Read only mode.
- **MyTemplates**, in Read - Write mode.
- **Plugin Templates**, in Read only mode.

2.2. Refresh

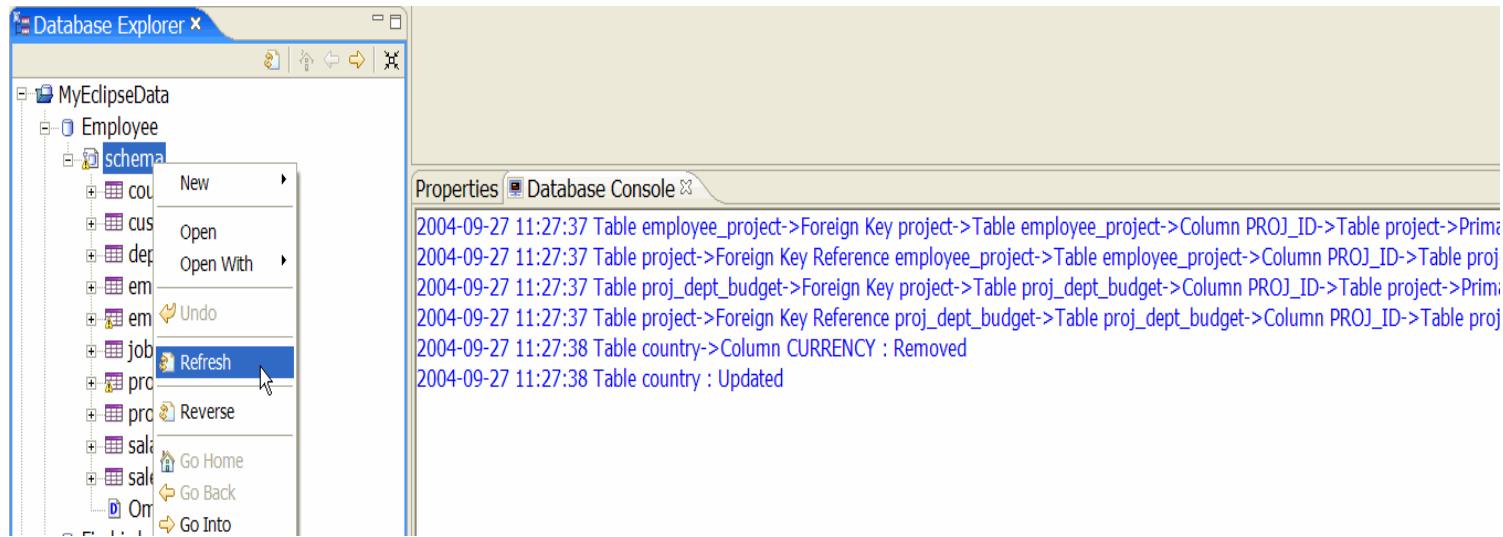


The **Refresh** command re-reads the file content.

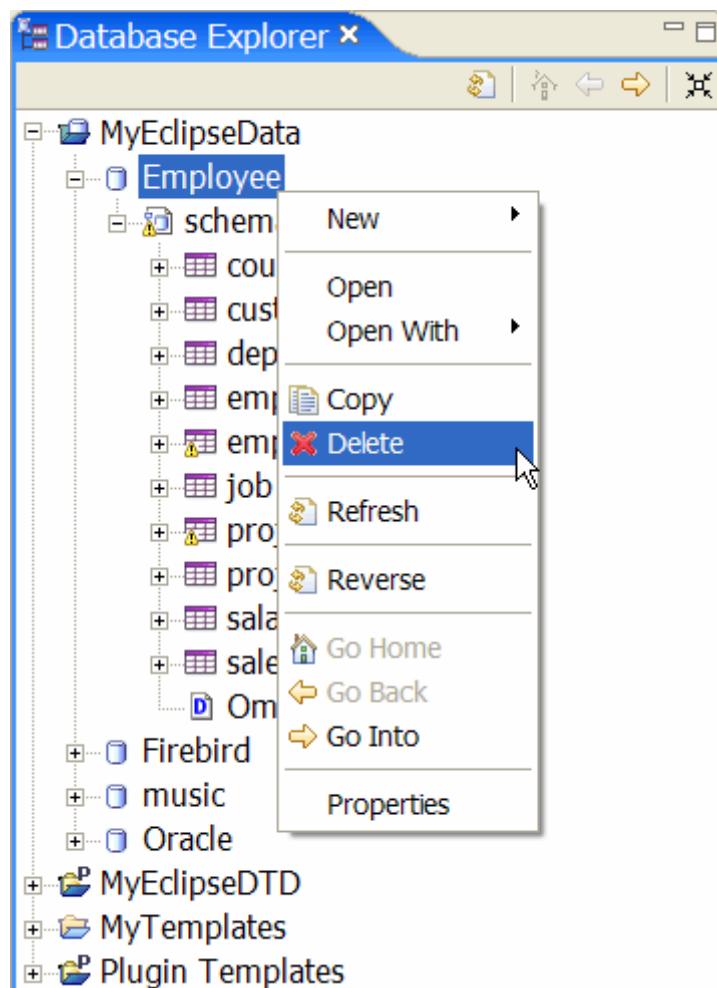
If a **Database Connection** is re-read, no merge process is performed.



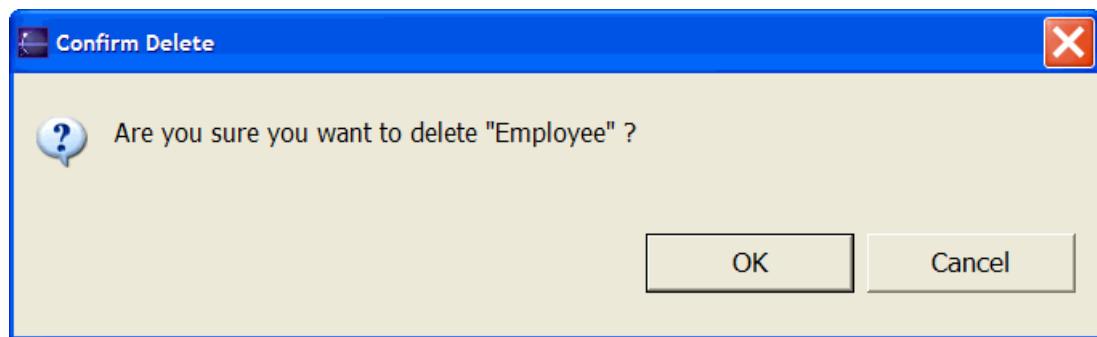
if a **Database Schema** is re-read, a merge process is performed.



2.3. Delete

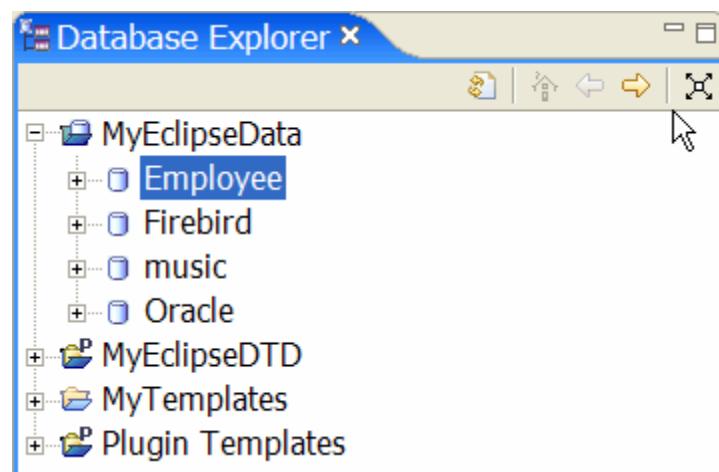
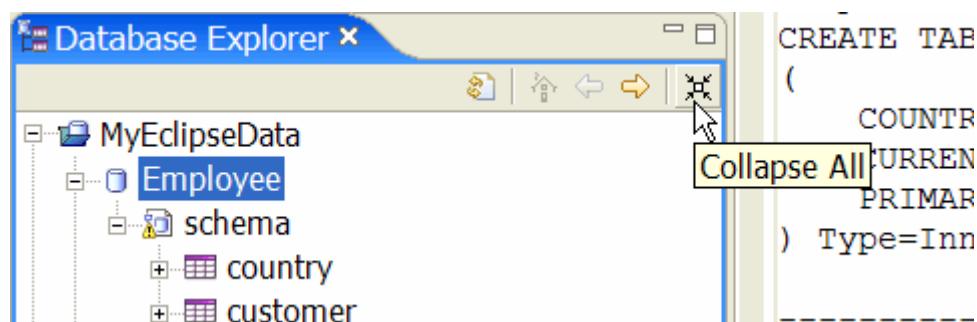


Database Connection files can be deleted.
A dialog will ask you to confirm your deletion.

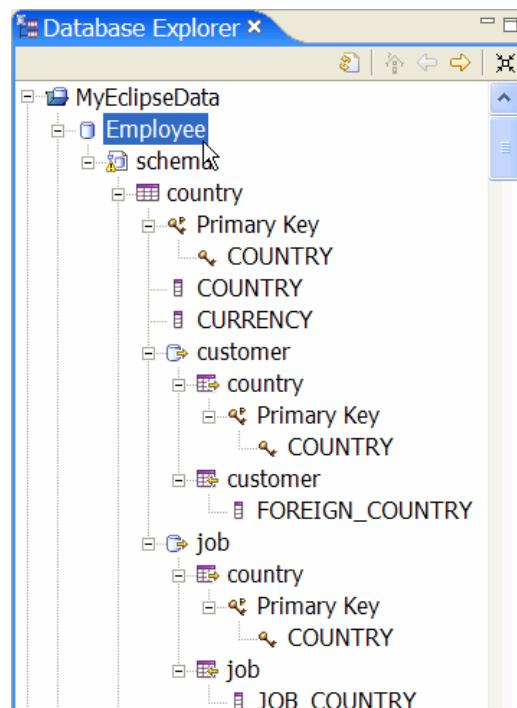
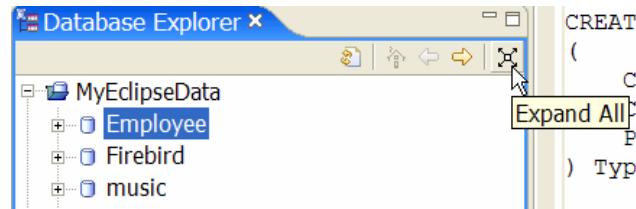


2.4. Collapse All and Expand All

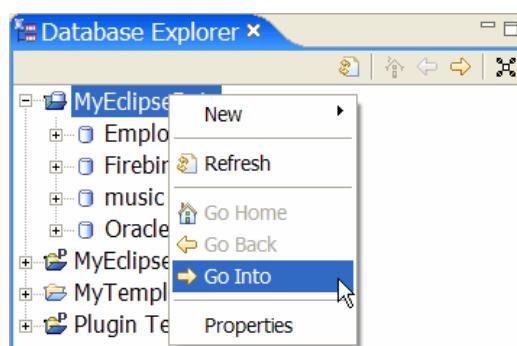
The **Collapse All** command collapses the current node.

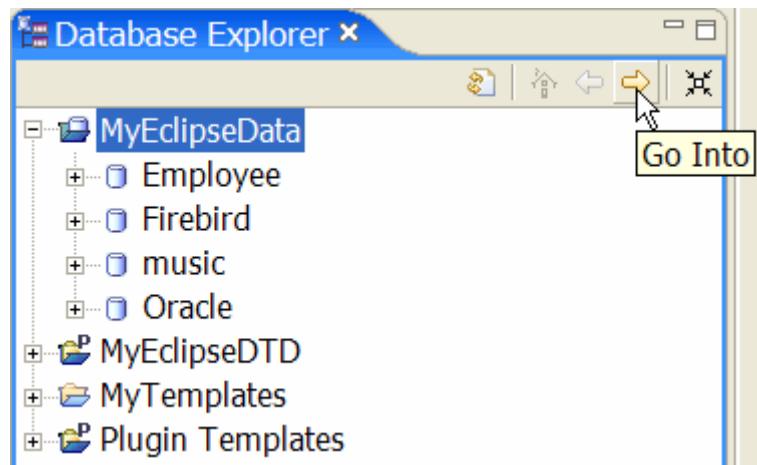


The **Expand All** command expands the current node.

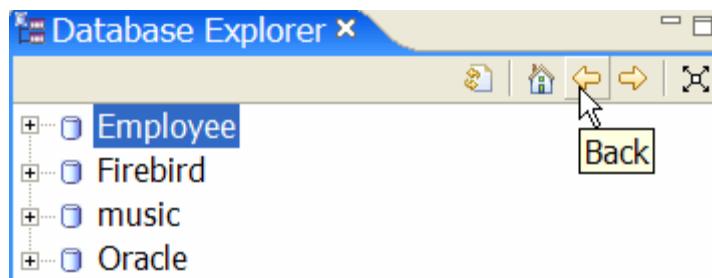


2.5. Home, Go Home, Back, Go Back and Go Into

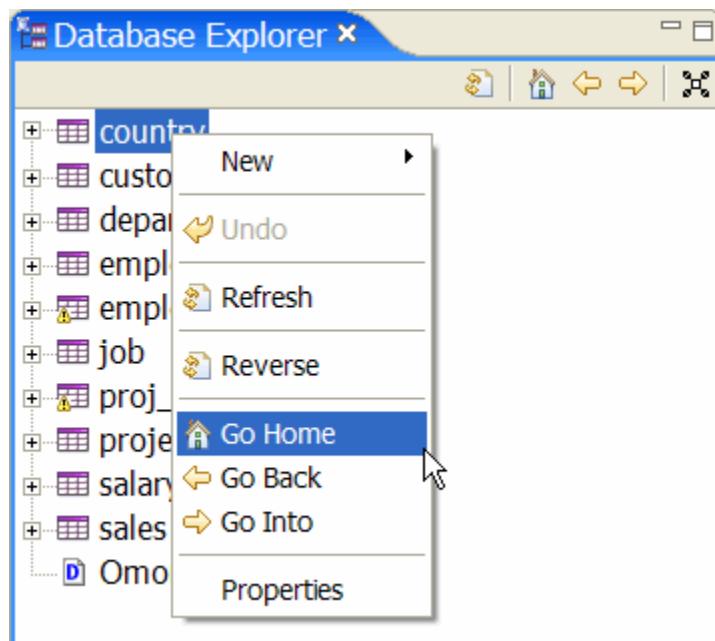




The **Go Into** command lets you zoom in to a particular node of your tree.



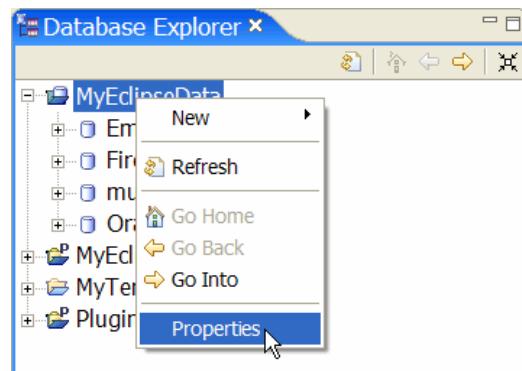
The **Back** command lets you zoom out to the current node of your tree.



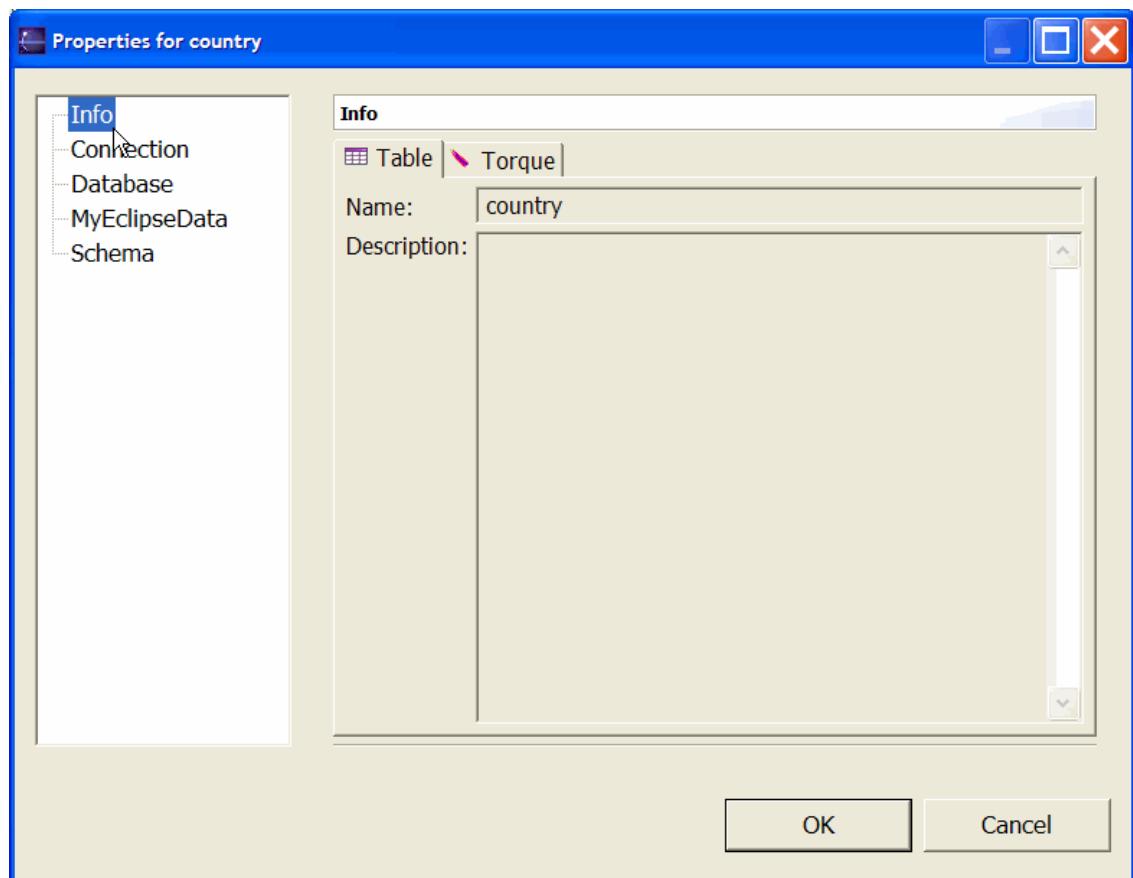
While the **Go Home** command lets you go to the root node of your tree.

2.6. Properties

Each node of your tree has a **Properties** command.



This command opens a properties page of your current node.



3. Properties View

Properties		Database Console
Property	Value	
Connection		
+ Classpath		
+ JDBC		
last modified	24/09/04 17:34	
location	D:\Outils\eclipse-3.0.1-Test\workspace\.metadata\.plugins\com.omondo.c	
name	Employee	
path	./database/Employee/Employee.odc.xml	
Database		
last modified	24/09/04 15:14	
location	D:\Outils\eclipse-3.0.1-Test\workspace\.metadata\.plugins\com.omondo.c	
name	Employee	
path	./database/Employee	
MyEclipseData		
last modified	24/09/04 15:14	
location	D:\Outils\eclipse-3.0.1-Test\workspace\.metadata\.plugins\com.omondo.c	
name	.database	
path	./database	
Schema		
default foreign key on delete	none	
default foreign key on update	none	
last modified	27/09/04 11:29	
location	D:\Outils\eclipse-3.0.1-Test\workspace\.metadata\.plugins\com.omondo.c	

The **Window -> Show View -> Properties** will open a specialized properties view in your current perspective.

This view displays the current set of properties of a selected Eclipse platform object.

This object could be a resource in your Navigator View, a Java object in your Package Explorer, an opened editor or a selected node in your DatabaseExplorer.

Database Connection

1. Introduction
2. New Database Connection
 1. New Database Connection
 1. Database Name
 2. User ID
 3. Password
 4. JDBC Connection URL
 5. Default Value
 6. Check Database Connection
 7. Meta Data
 1. Schema Name
 2. Schemas
 2. New Database Schema
 1. Database Schema Name
 3. New Database Schema DTD
 1. Database Schema DTD Name
 4. Finish
3. Database Connection Update
4. Database Schema Update
5. Database Schema DTD Update
6. Reverse Command

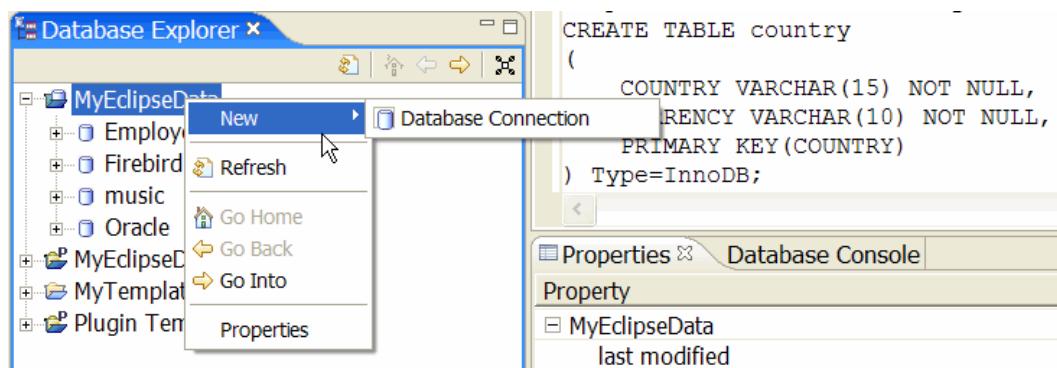
1. Introduction

The purpose of this chapter is to show you how to create and update a Database Connection in the Database Explorer.

2. New Database Connection

Select a MyEclipseData node, right-click and select :

New -> Database Connection.



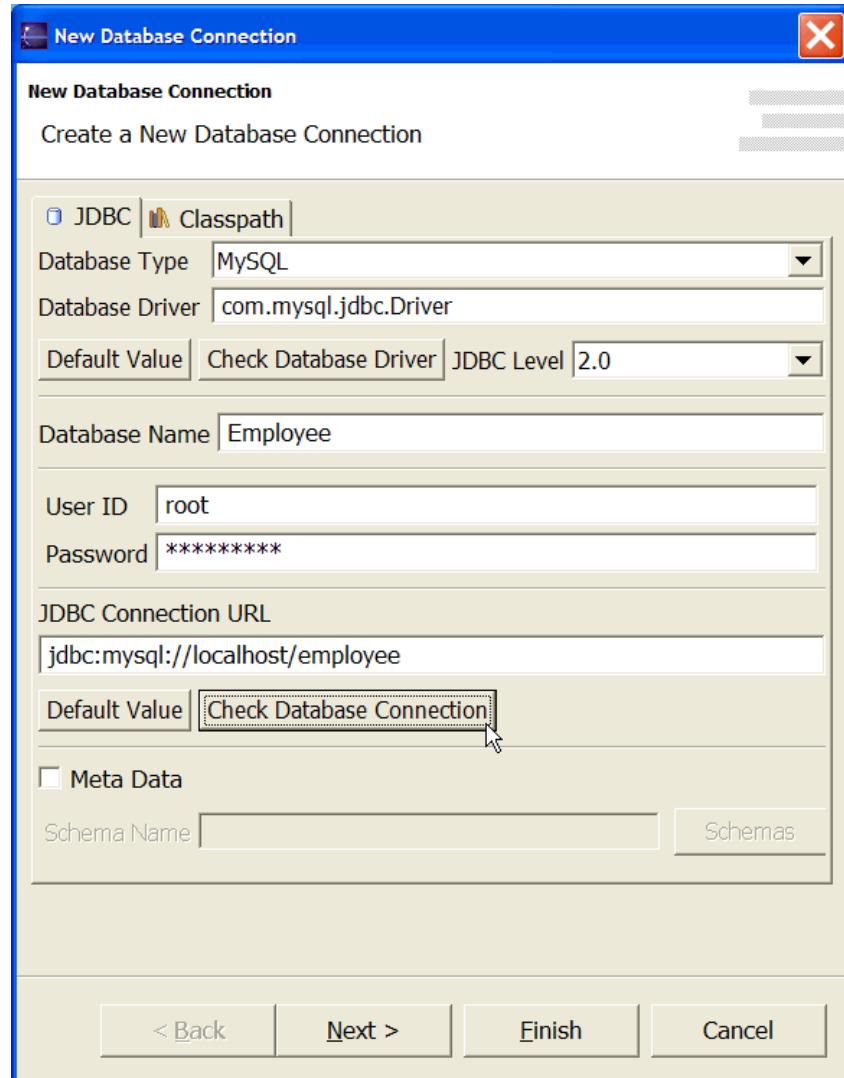
The New Database Connection wizard is composed of the following pages :

- New Database Connection
- New Database Schema
- New Database Shema DTD

2.1. New Database Connection

The Database Connection page inherits the [Connection Global Preferences](#).

The [Reverse](#) check box is not inherited as this wizard will attempt to retrieve the Meta Data of your connection.



2.1.1. Database Name

Database Name is the name of your new Database Connection file.

Each resource should have a unique name in the targeted selected workspace folder or project root.
This data is not related to any Meta Data informations.

This name will be the file name of the generated Database Connection file.

A Database Connection is an XML file, its file extension could be :

- .odc.xml
- .odc

This file extension is managed by the [Save Policy](#) Preference.

2.1.2. User ID

User ID is your JDBC user id.

User ID is stored in your Database Connection file.

2.1.3. Password

Password is your JDBC password.
Password is stored in your Database Connection file.

2.1.4. JDBC Connection URL

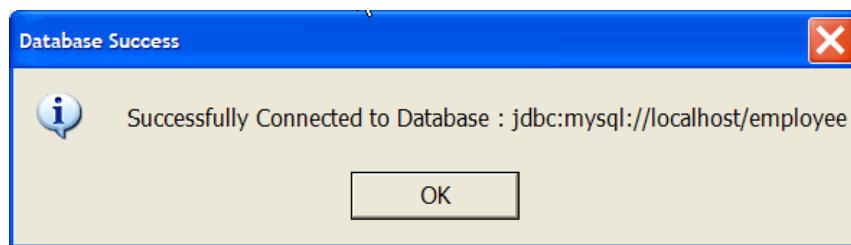
The JDBC Connection URL described the JDBC connection information.
Each time you select another Database Type, a default JDBC URL is displayed.
This field is updatable.

2.1.5. Default Value

The Default value reset the JDBC Connection URL to its predefined value.

2.1.6. Check Database Connection

Click on the Check Database Driver and EclipseDatabase will attempt to connect your Database.
Your User ID, Password and Connection URL will then be used.
If you have a successful connection :



If you have an unsuccessful connection

Database Success

Successfully Connected to Database : jdbc:mysql://localhost/employee

OK

JDBC | Classpath |

Database Type : MySQL

Database Driver : com.mysql.jdbc.Driver

Default Value | Check Database Driver | JDBC Level : 2.0

Database Name : Employee

User ID : root

Password : *****

JDBC Connection URL : jdbc:mysql://localhost/employee

Default Value | Database Error

Meta Data | Schema Name |

OK

< Back | Next > | Finish | Cancel

ADDRESS T.TNE1 VARCHAR(30) .

Properties | Database Console

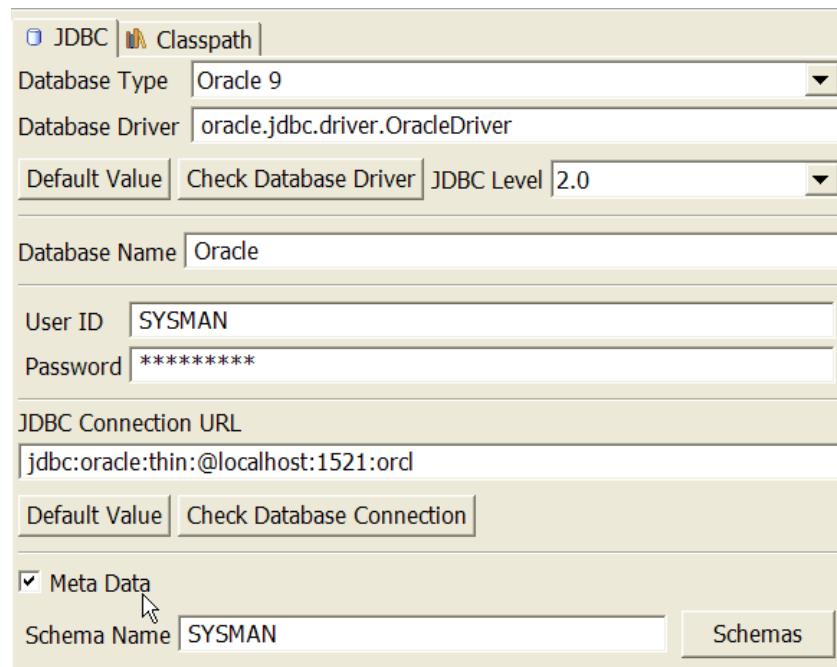
2004-09-28 10:45:16 Couldn't load JDBC driver com.mysql.jdbc.Driver
2004-09-28 10:45:16 com.mysql.jdbc.Driver
2004-09-28 10:45:38 Couldn't connect to database :
2004-09-28 10:45:38 Couldn't connect to database : Invalid authorization specification message from server: "Access denied"

You can find more information about what happened in the [DatabaseConsole](#).

However check the following :

- [Driver Settings](#)
- [User ID](#)
- [Password](#)
- [JDBC Connection URL](#)

2.1.7. Meta Data



The Meta Data check box activates the Schema Meta Data selector.

Your Database Meta Data will then be narrowed with this Schema value.

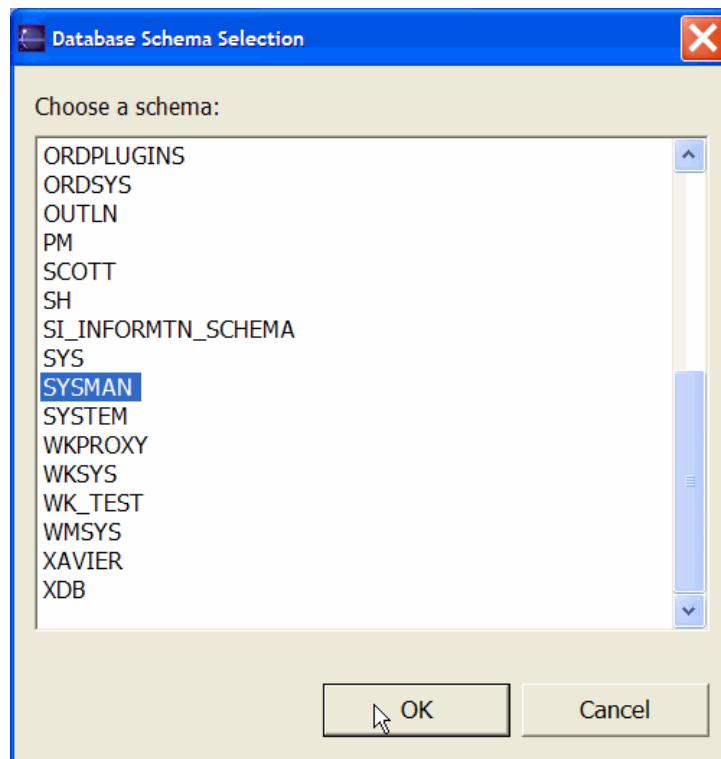
2.1.7.1. Schema Name

Schema Name is a Database Meta Data Schema name.

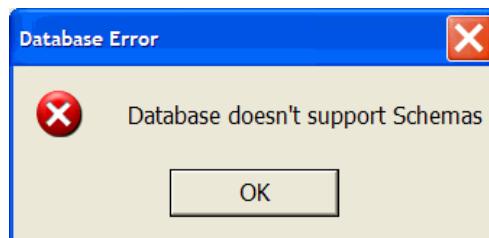
This value will narrow your reversed Meta Data.

2.1.7.2. Schemas

This button activates a Database Meta Data Schemas selector.

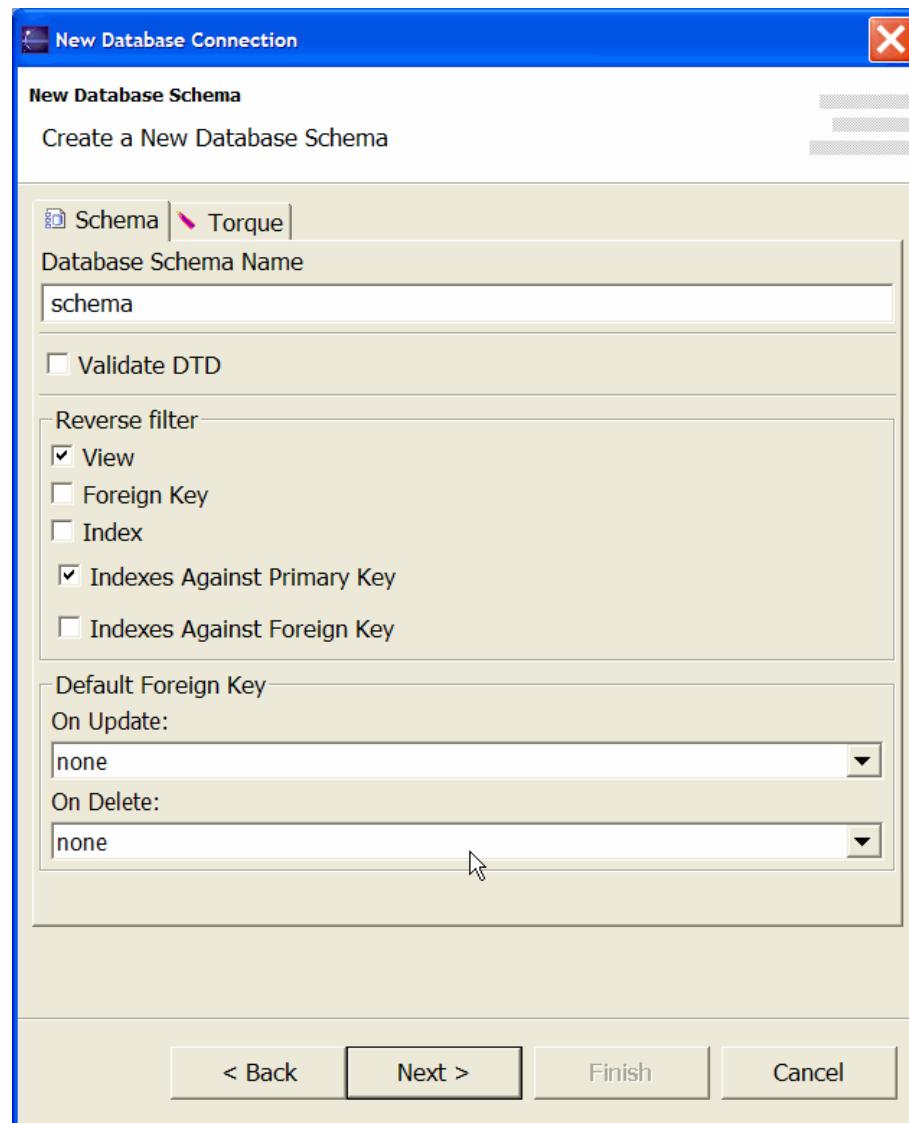


If your current database doesn't support Meta Data Schemas, an error will be displayed.



2.2. New Database Schema

The Database Schema page inherits the [Schema Global Preferences](#).



2.2.1. Database Schema Name

Database Schema Name is the name of your new Database Schema file.

Each resource should have a unique name in the targeted selected workspace folder or project root.
This data is not related to any Meta Data informations.

This name will be the file name of the generated Database Schema file.

A Database Schema is an XML file, its file extension could be :

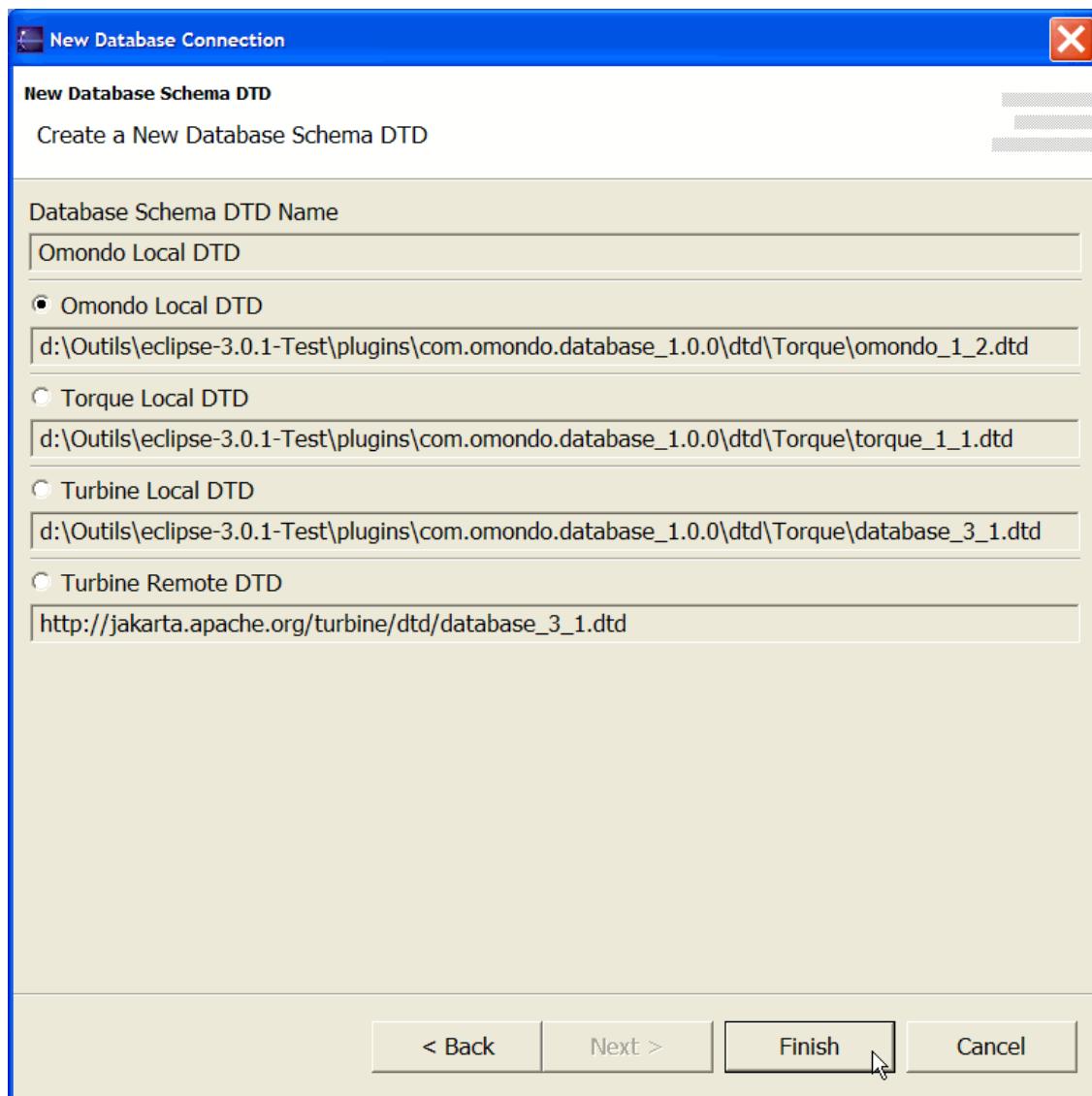
- .ods.xml
- .ods

This file extension is managed by the [Save Policy](#) Preference.

This file is referenced in your current [Database Connection](#).

2.3. New Database Schema DTD

The Database Schema DTD page inherits the [Schema DTD Global Preferences](#).



2.3.1. Database Schema DTD Name

Database Schema DTD Name is the name of your new Database Schema DTD file.

Each resource should have a unique name in the targeted selected workspace folder or project root.
This data is not related to any Meta Data informations.

This name will be the file name of the generated Database Schema DTD file.

A Database Schema DTD is an DTD file, its file extension is :

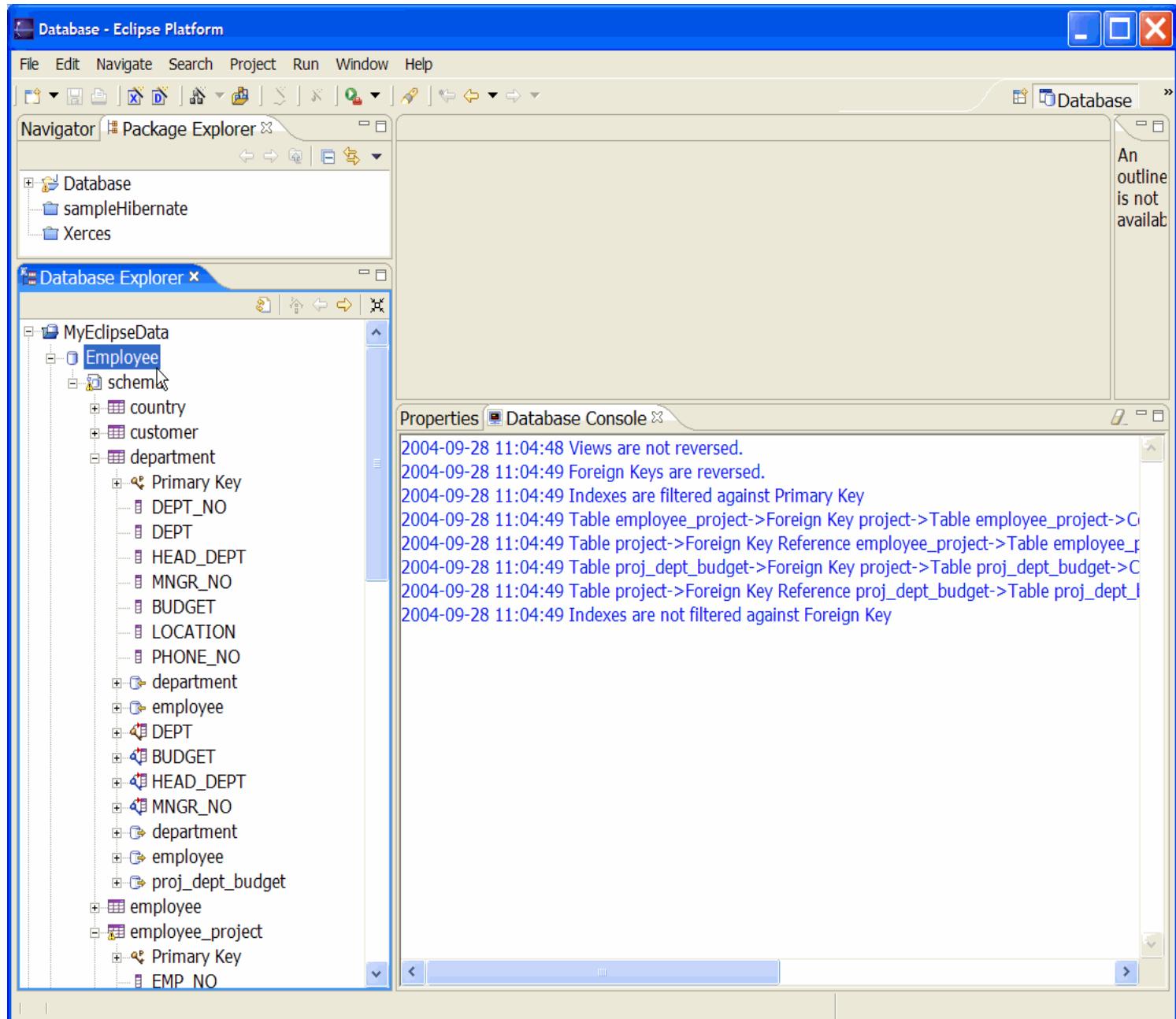
- .dtd

This file is referenced in your current [Database Schema](#).

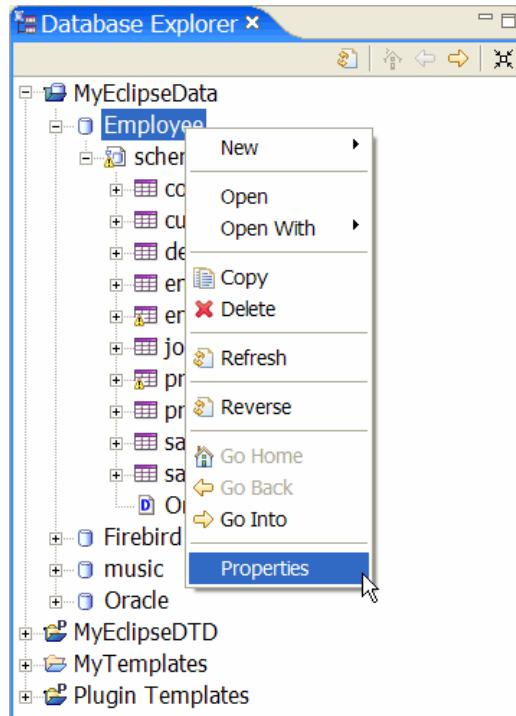
2.4. Finish

The reverse of the new Database Connection will be processed and displayed in the Database Explorer.

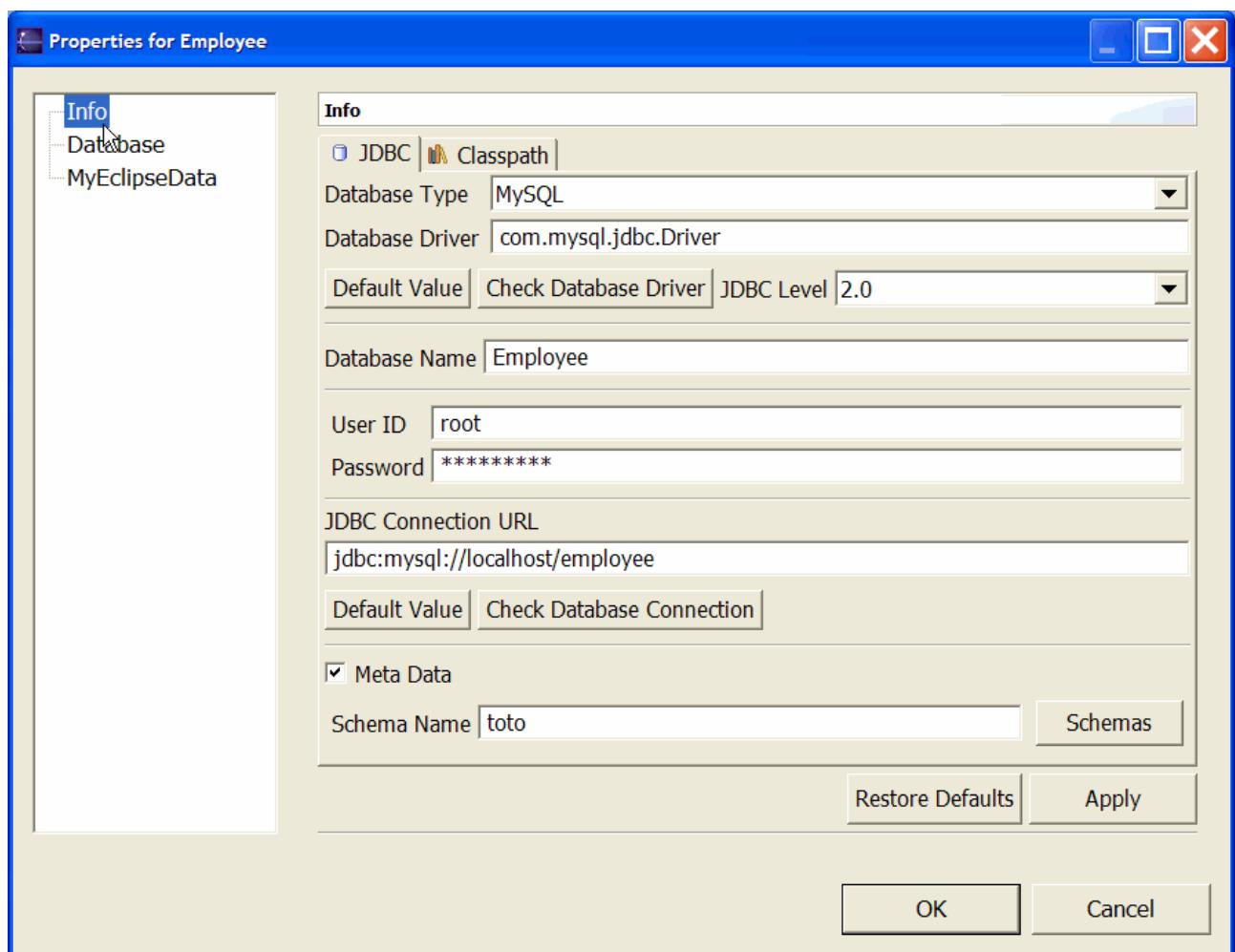
A detailed information will be output in the DatabaseConsole window.



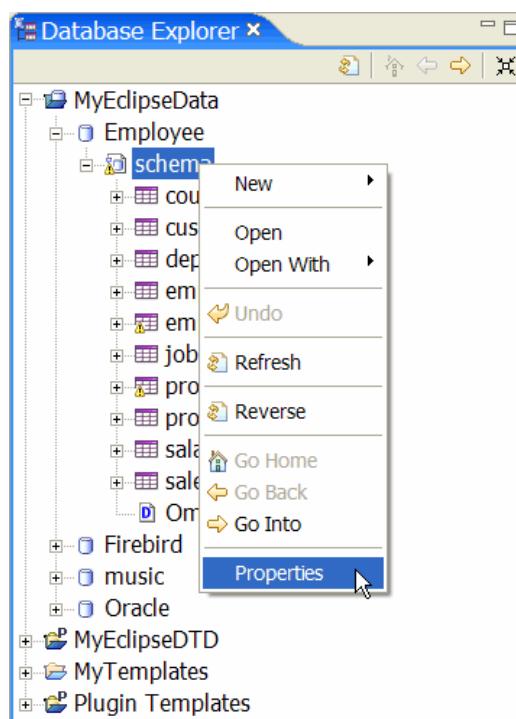
3. Database Connection Update



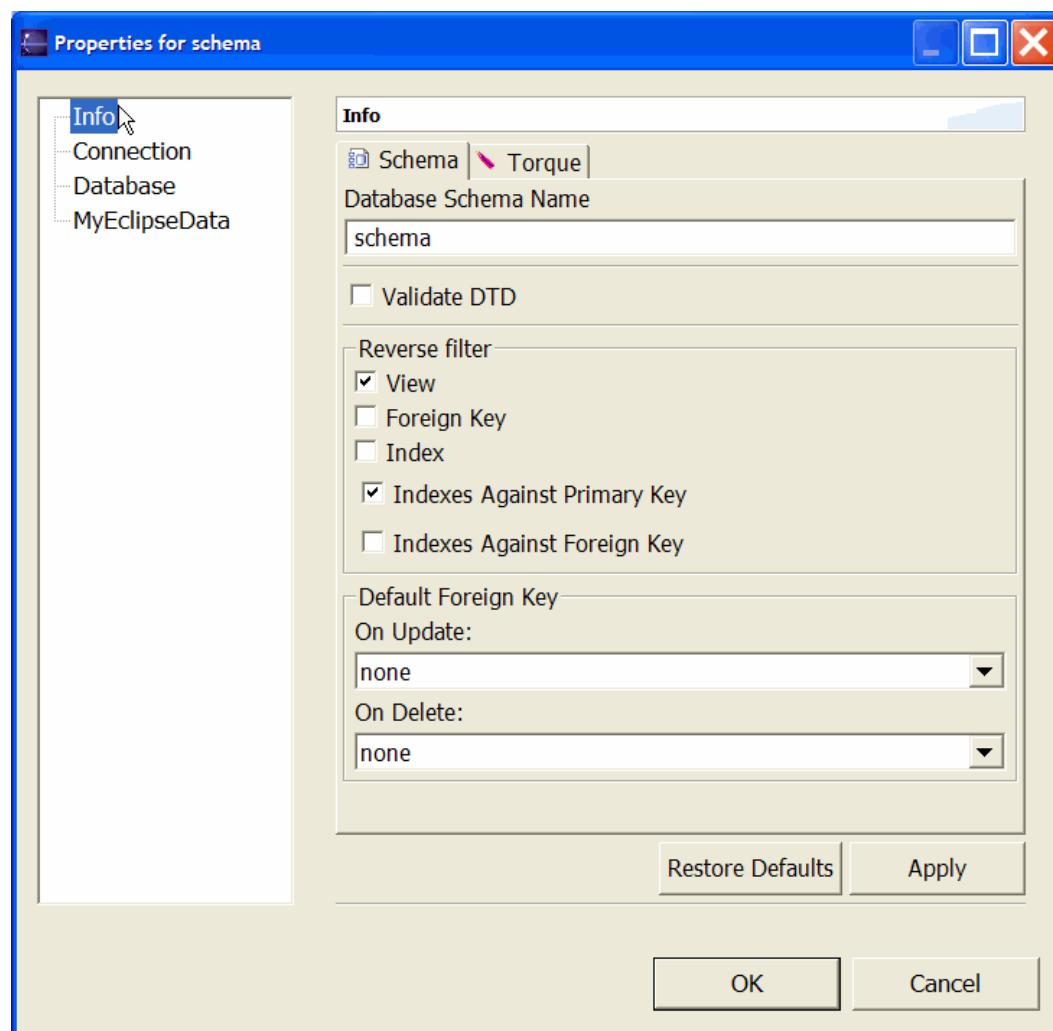
Any Database Connection could be updated.
Select a Database Connection node, right-click and select **Properties**.



4. Database Schema Update

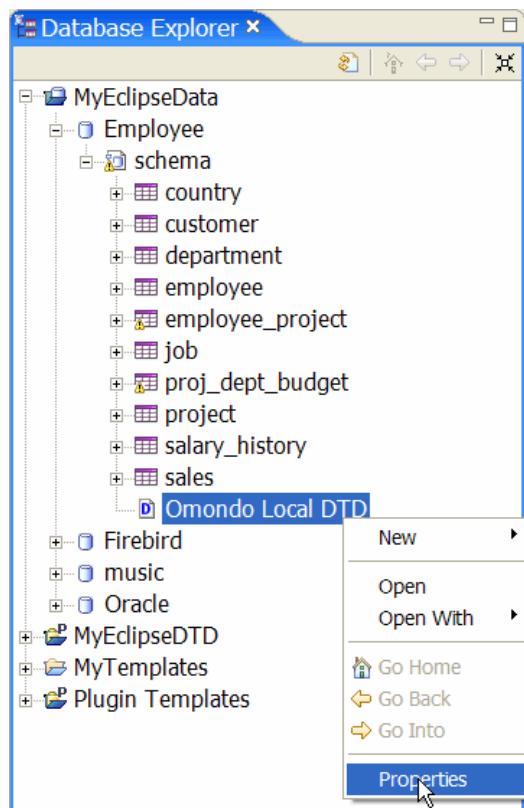


Any Database Schema could be updated.
Select a Database Schema node, right-click and select **Properties**.



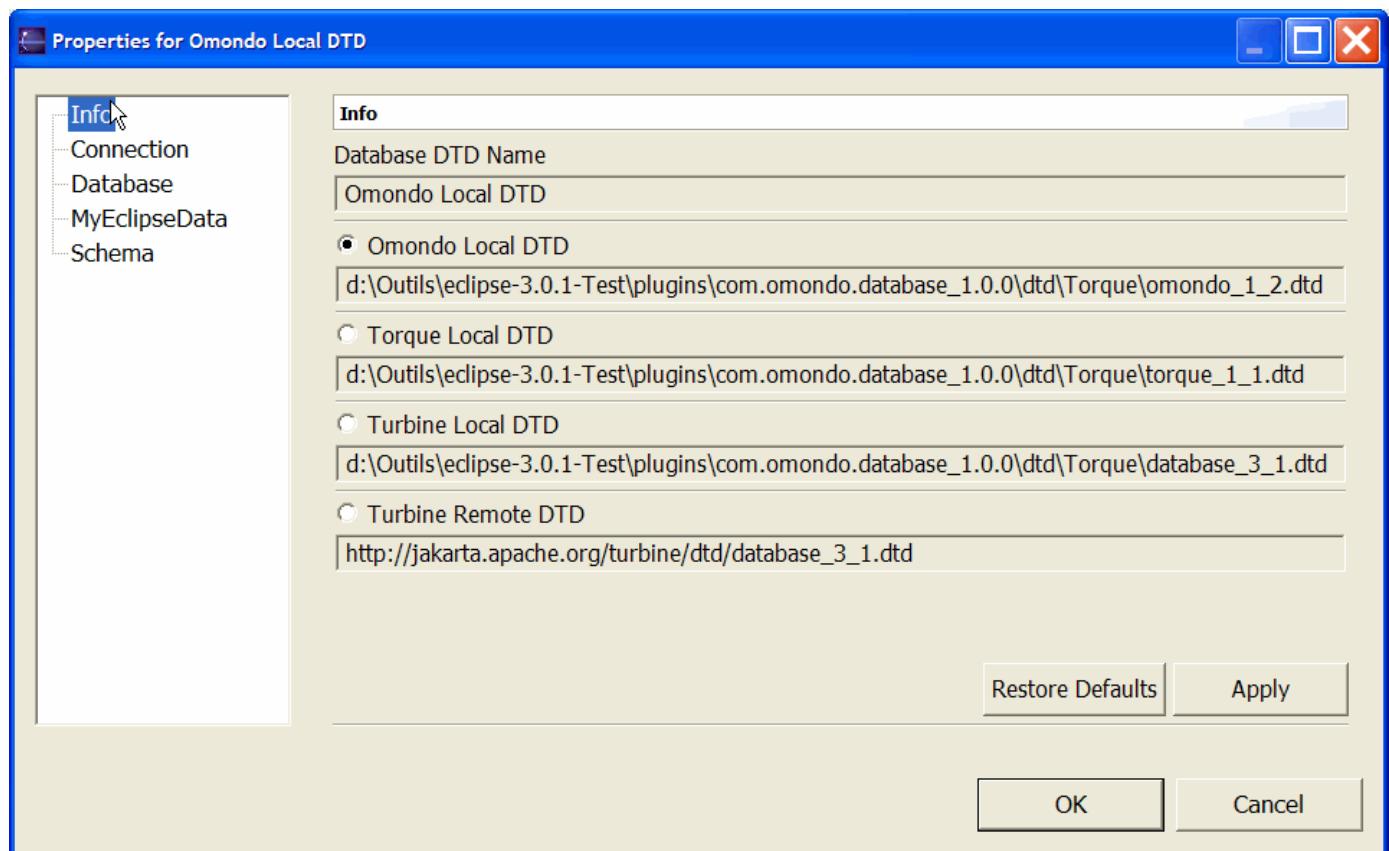
Parents informations are inherited.
It means that you can modify the Database Connection informations of a Database Schema.

5. Database Schema DTD Update



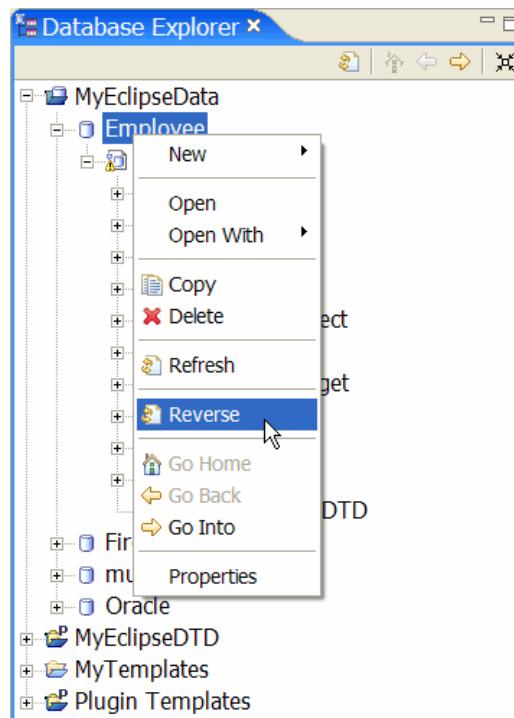
Any Database Schema DTD could be updated.

Select a Database Schema DTD node, right-click and select **Properties**.



Parents informations are inherited. It means that you can modify the Database Connection and the Database Schema informations of a Database Schema DTD.

6. Reverse Command



Select a Database Connection node, right-click and select **Reverse**.

The Database Meta Data will be reversed and a merge process will be performed with the current content of your Database Schema.

A report will be displayed in the DatabaseConsole.

This screenshot shows the Database Explorer and Database Console windows side-by-side. The Database Explorer on the left shows the 'Employee' schema expanded, revealing tables like 'artist', 'cd', 'cd_track', 'lyrics', and 'track'. The Database Console on the right displays a log of operations performed during the reverse process. The log entries are:

```

2004-09-28 12:24:30 Views are not reversed.
2004-09-28 12:24:30 Foreign Keys are reversed.
2004-09-28 12:24:30 Indexes are filtered against Primary Key
2004-09-28 12:24:30 Indexes are not filtered against Foreign Key
2004-09-28 12:24:30 Table sales : Removed
2004-09-28 12:24:30 Table salary_history : Removed
2004-09-28 12:24:30 Table project : Removed
2004-09-28 12:24:30 Table proj_dept_budget : Removed
2004-09-28 12:24:30 Table job : Removed
2004-09-28 12:24:30 Table employee_project : Removed
2004-09-28 12:24:30 Table employee : Removed
2004-09-28 12:24:30 Table department : Removed
2004-09-28 12:24:30 Table customer : Removed
2004-09-28 12:24:30 Table country : Removed
2004-09-28 12:24:30 Table artist->Column name : Added
2004-09-28 12:24:30 Table artist->Column age : Added
2004-09-28 12:24:30 Table artist->Primary Key : Added
2004-09-28 12:24:30 Table artist->Primary Key->Column name : Added
2004-09-28 12:24:30 Table artist : Added
2004-09-28 12:24:30 Table cd->Column name : Added
2004-09-28 12:24:30 Table cd->Column price : Added
2004-09-28 12:24:30 Table cd->Primary Key : Added
2004-09-28 12:24:30 Table cd->Primary Key->Column name : Added
2004-09-28 12:24:30 Table cd : Added
2004-09-28 12:24:30 Table cd_track->Column cd_name : Added
2004-09-28 12:24:30 Table cd_track->Column track_id : Added
2004-09-28 12:24:30 Table cd_track->Primary Key : Added
2004-09-28 12:24:30 Table cd_track->Primary Key->Column cd_name : Added

```

Modeling in the Workspace

1. [Introduction](#)
2. [Drag and Drop](#)
3. [Database Connection](#)
4. [Database Create SQL Script](#)
5. [Database Diagram](#)
6. [Database Schema SQL Script](#)
7. [Database Data DTD and XML Resources](#)
8. [Forward](#)
9. [Index](#)

1. Introduction

You will learn in this chapter how to work with Databases inside the Eclipse Workspace. EclipseDatabase proposes several wizards who will help you to manage your database work.

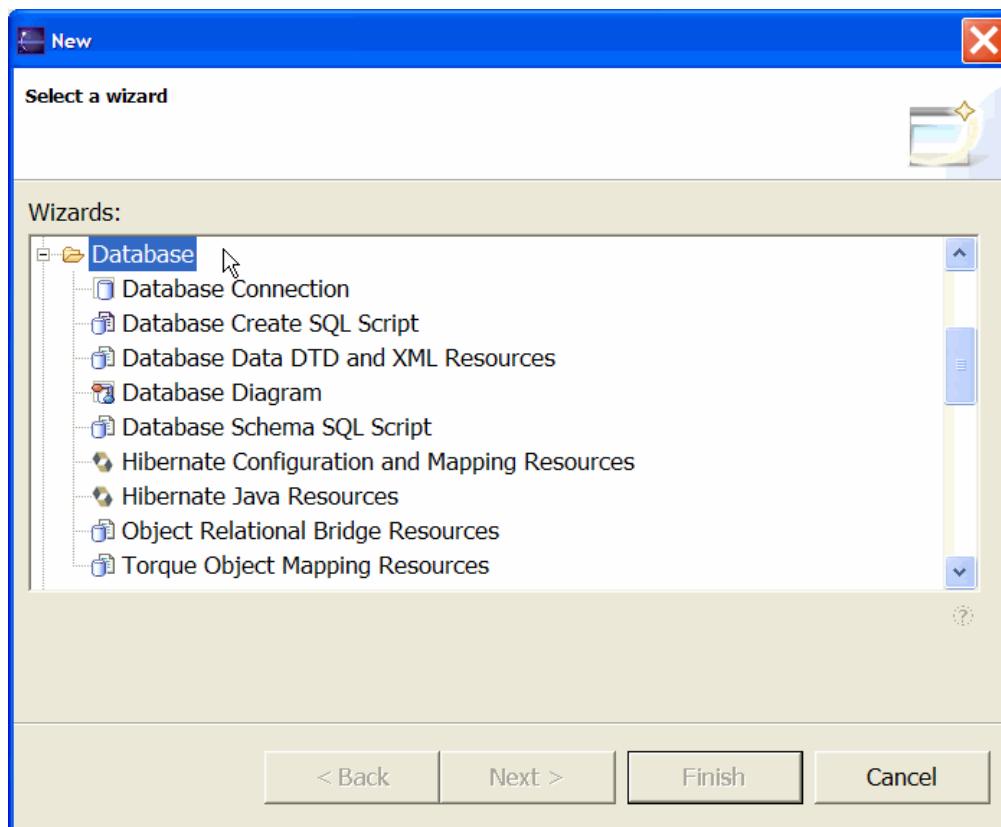
- Database Connection
- Database Create SQL Script
- Database Diagram
- Database Schema SQL Script
- Database Data DTD and XML Resources

There are several ways to run a wizard.

File->New->Other->Database

or

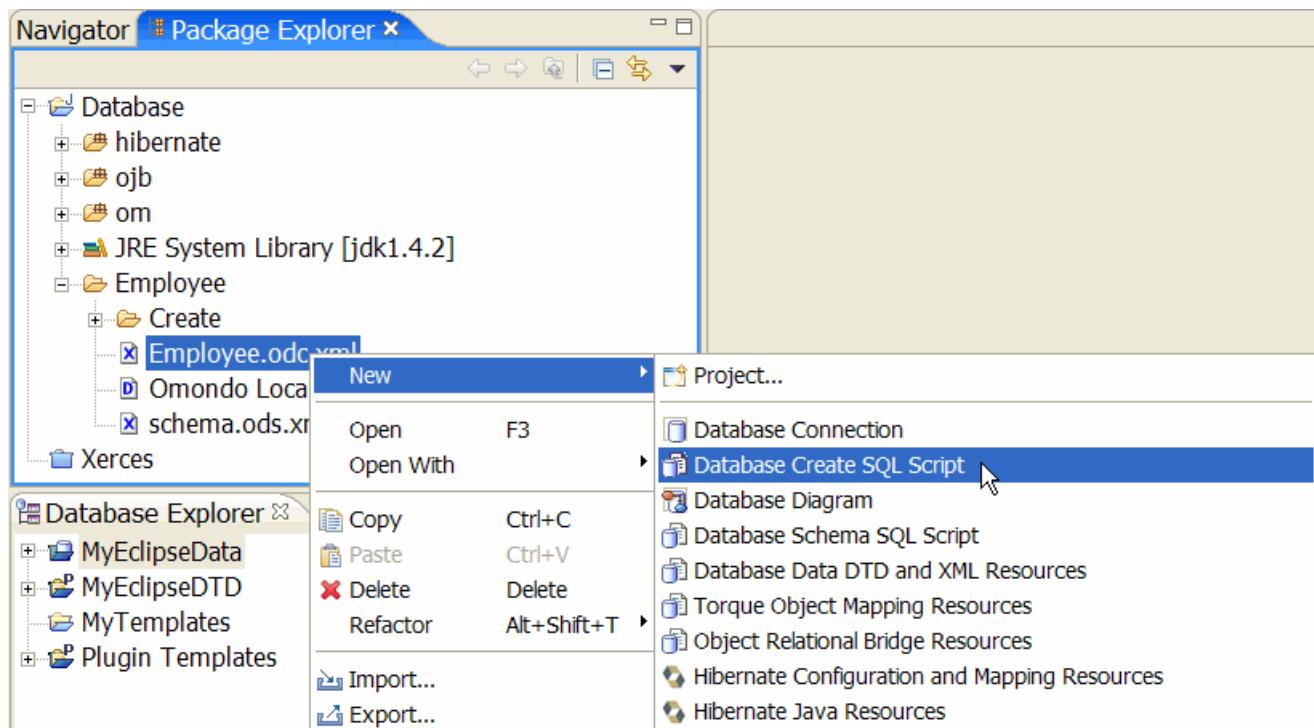
Select an object in the Package Explorer View or the Navigator View then right-click and select : **New->Other->Database**



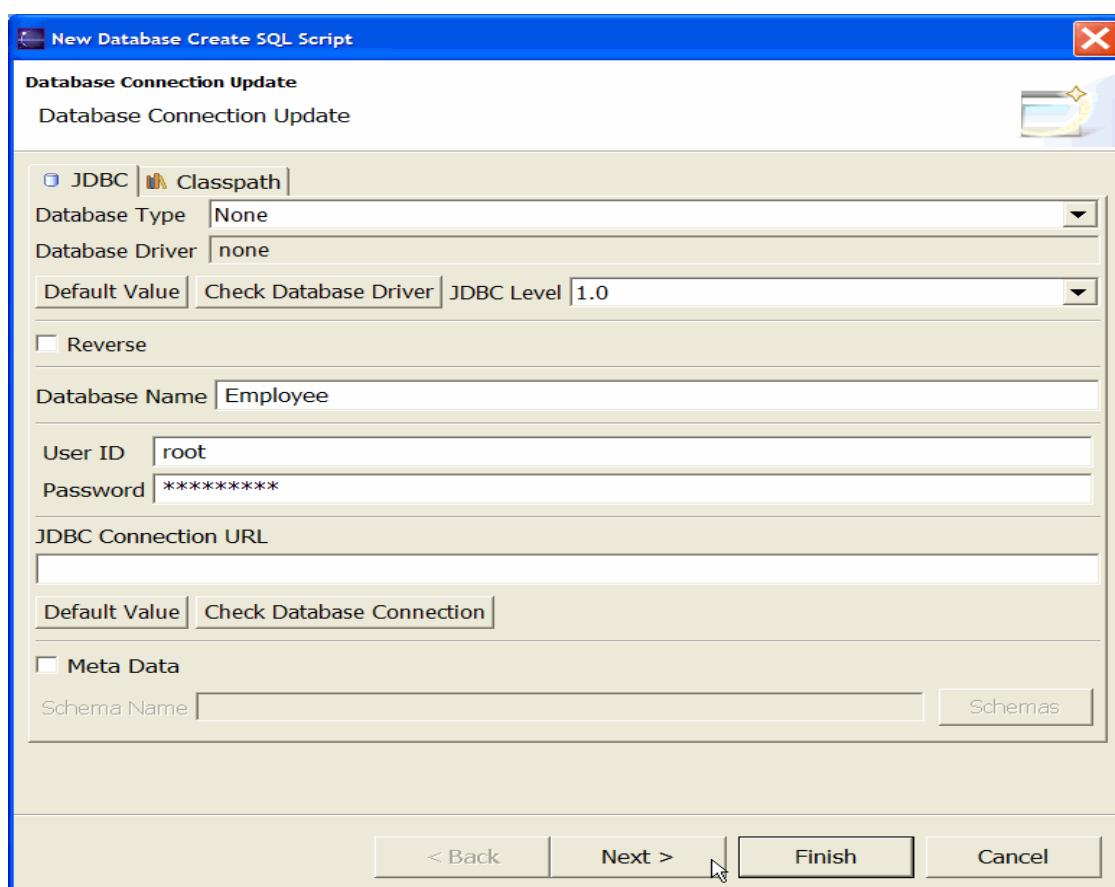
The difference between this two ways of running a wizard is the context. Everytime you start a wizard, EclipseDatabase knows where you come from. For example here is the exact behaviour of the following wizards :

- Database Create SQL Script
- Database Diagram
- Database Schema SQL Script
- Database Data DTD and XML Resources

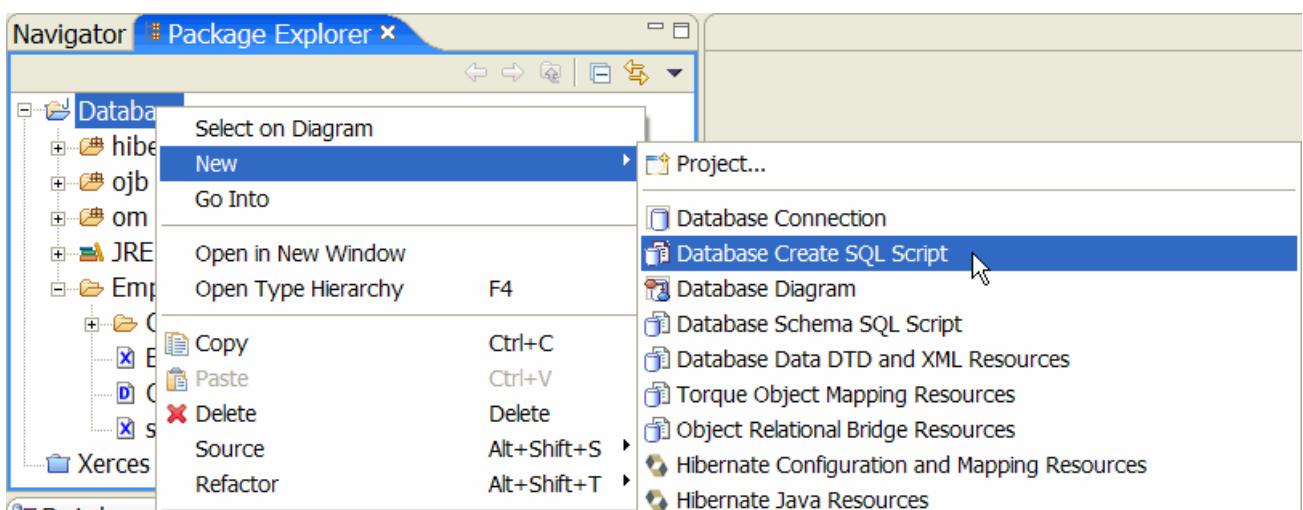
If you start one of these wizards while selecting a Database Connection file, a Database Schema file or any file belonging to a Database Connection, EclipseDatabase will pre-set this connection:



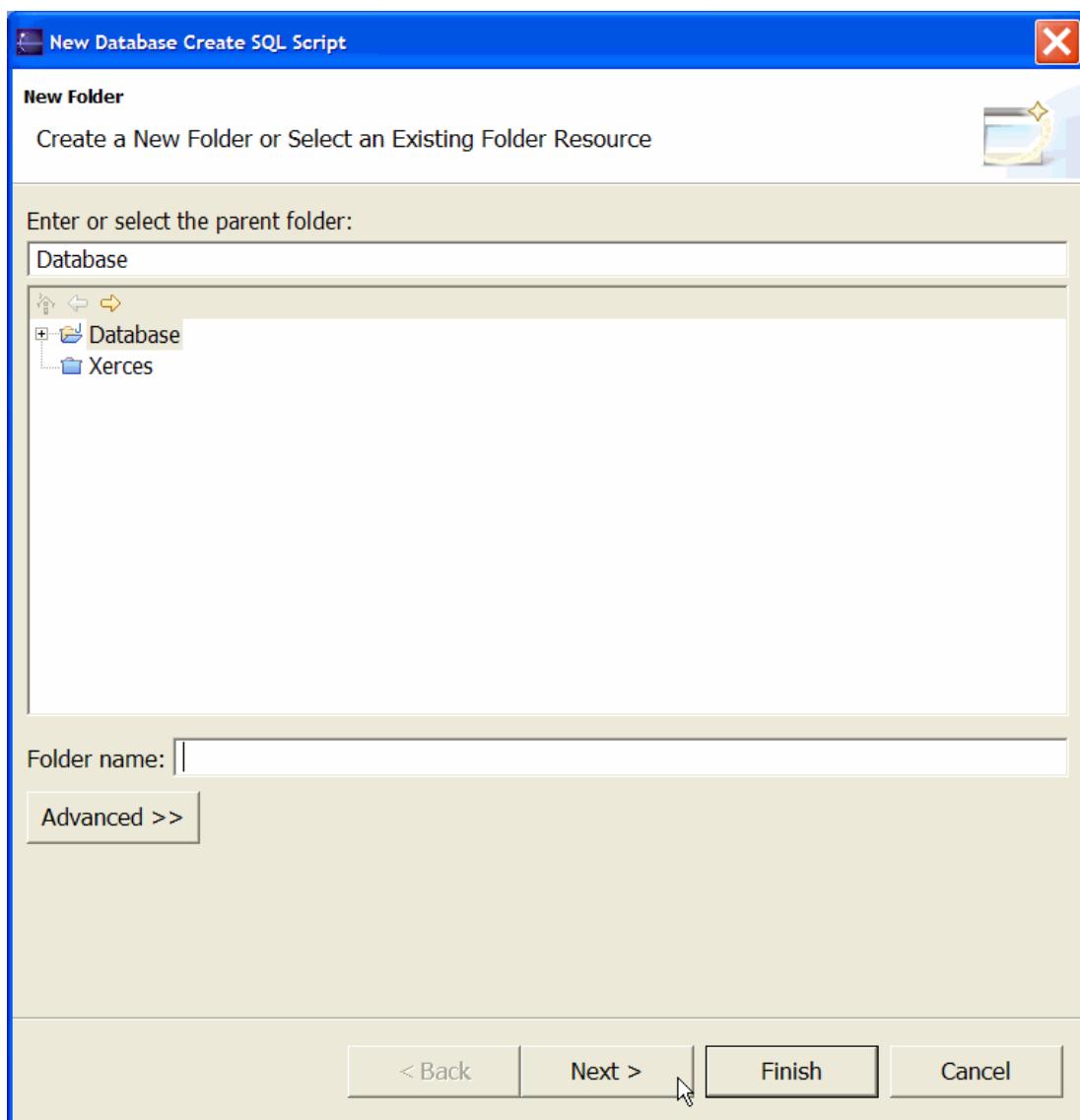
This will open a new Database Create SQL Script pre-set to your selected Database Connection



Otherwise:



This will open a new Database Create SQL Script with a target selector.



This way of working means that you can modify a current connection (Connection, Schema and DTD) while you fill the target of a specific wizard.

To analyse the context EclipseDatabase use an [Index](#) to retrieve the Database information.

Drag and Drop

1. [Introduction](#)
2. [Drag and Drop](#)

1. Introduction

The Drag and Drop mechanism lets you transfer an existing Database Connection from the [Database Explorer](#) to an existing Project inside your Eclipse Workspace.

This Project could be a Simple Project, a Java Project or another kind of Project supported by your Eclipse Platform.

With the Drag and Drop feature you will transfer three files inside your workspace :

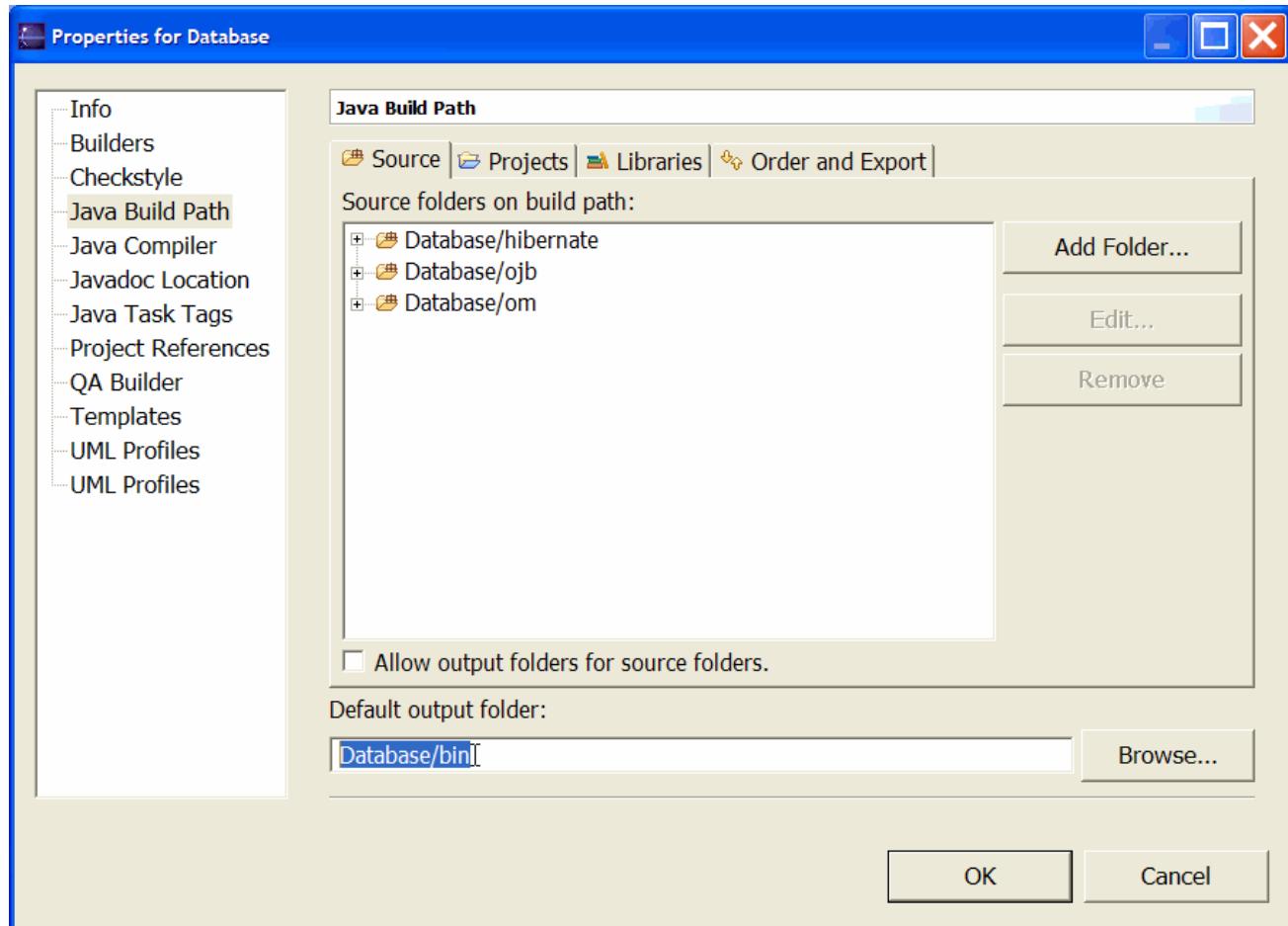
- A [Database Connection](#) file
- A [Database Schema](#) file
- A [Database Schema DTD](#) file

2. Drag and Drop

You should drop a Database Connection in the Navigator View.

Due to a restriction in the Eclipse Platform, see [Eclipse Bugzilla Bug 71065](#), you cannot drop a Database Connection in the Package Explorer View.

If you are targeting a Java Project, do not drop a Database Connection into a **Default output folder**.



This folder hosts all the compilation results.

If you create a Database Connection in a Java Source folder, this Connection will be copied into this folder.

EclipseDatabase knows how to deal with such situations but we do not recommend this way of working.

Select a Database Connection in your DatabaseExplorer.

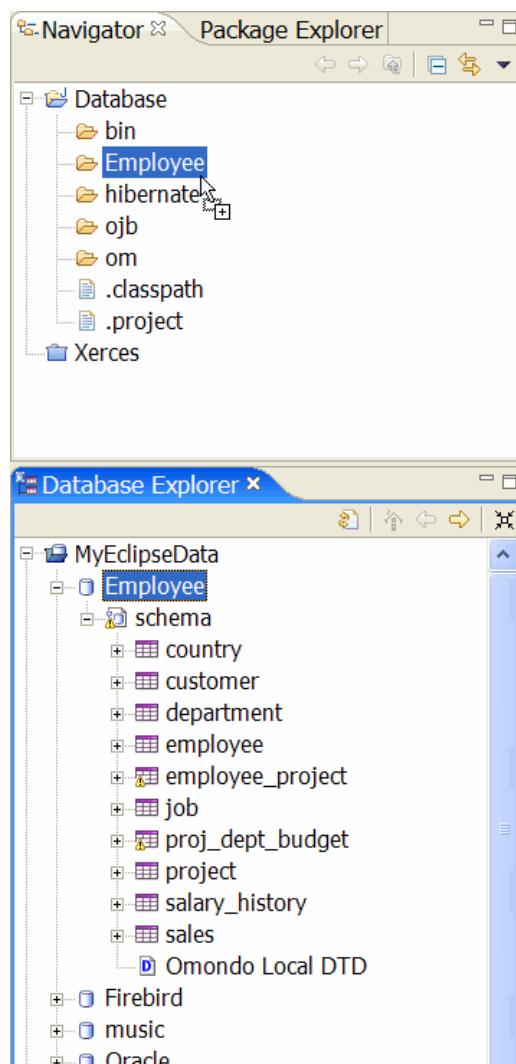
Keep your finger on the button.

Drag over a target in the Navigator View.

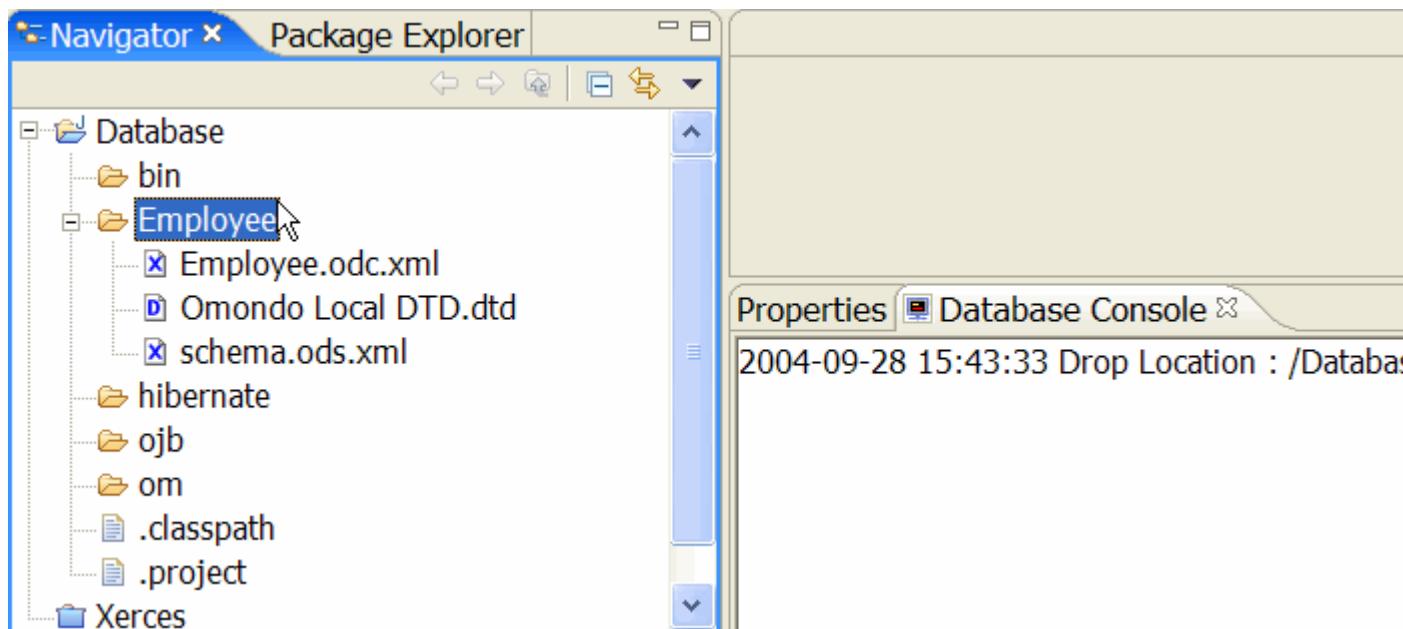
This could be a folder, a linked folder or a project root.

When the mouse button is released, the Database Connection has moved from the DatabaseExplorer to the Navigator window.

The following example shows how to move the Employee Database using the mouse :

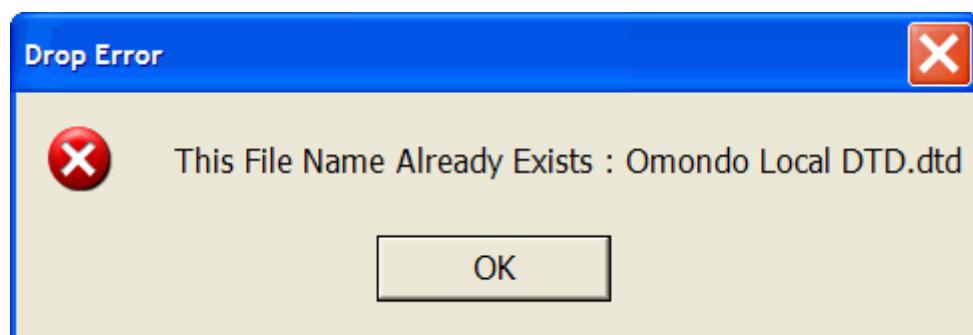


The DatabaseConsole shows a report of what happened during this move.



Occasionally Drag and Drop can fail.

In this case a dialog box is opened with an explanatory message.



Database Connection

1. Introduction
2. New Database Connection
 1. New Folder
 1. Parent Folder
 2. Folder name
 3. Advanced
 2. New Database Connection
 3. New Database Schema
 4. New Database Schema DTD
 5. Finish
3. Database Connection Update
 1. Database Connection Update
 2. Database Schema Update
 3. Database Schema DTD Update
4. Refresh
5. Reverse

1. Introduction

The purpose of this chapter is to show you how to create and update a Database Connection in the workspace.

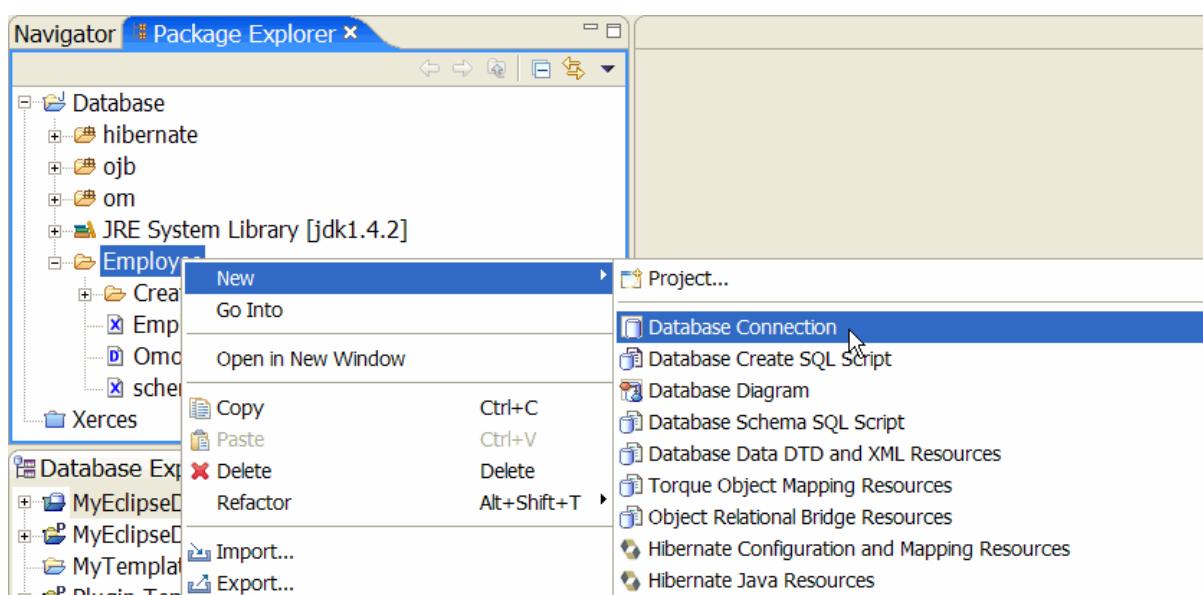
2. New Database Connection

To start the Database Connection wizard, select :

File->New->Other->Database->Database Connection

or

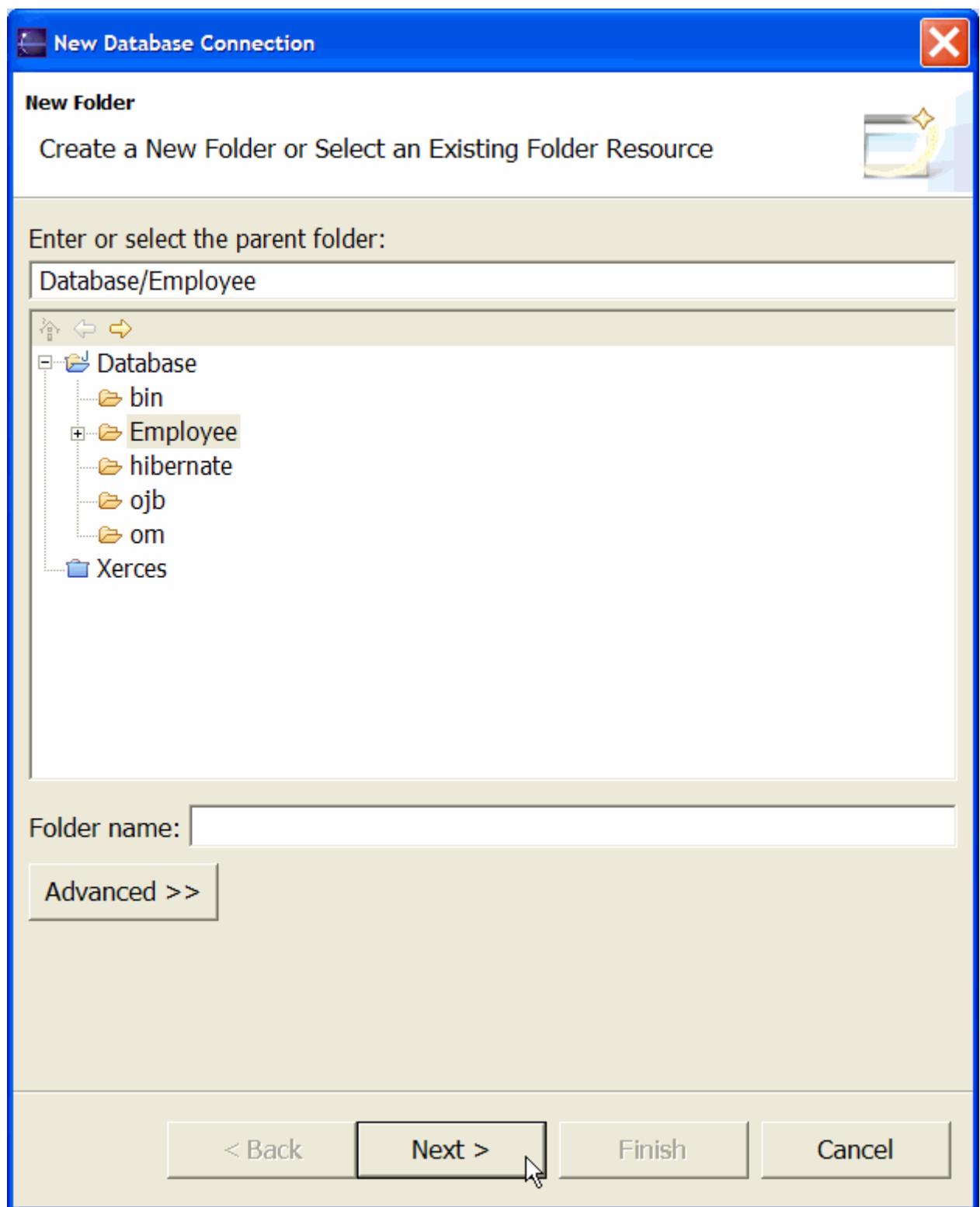
Select an object in the Package Explorer View or the Navigator View then right-click and select :
New->Other->Database->Database Connection



The New Database Connection wizard is composed of the following pages :

- New Folder
- New Database Connection
- New Database Schema
- New Database Schema DTD

2.1. New Folder



The New Folder page lets you choose a target inside your workspace.

2.1.1. Parent Folder

You can type an existing path or you can select an existing node in the resource tree. This selection is mandatory.

2.1.2. Folder name

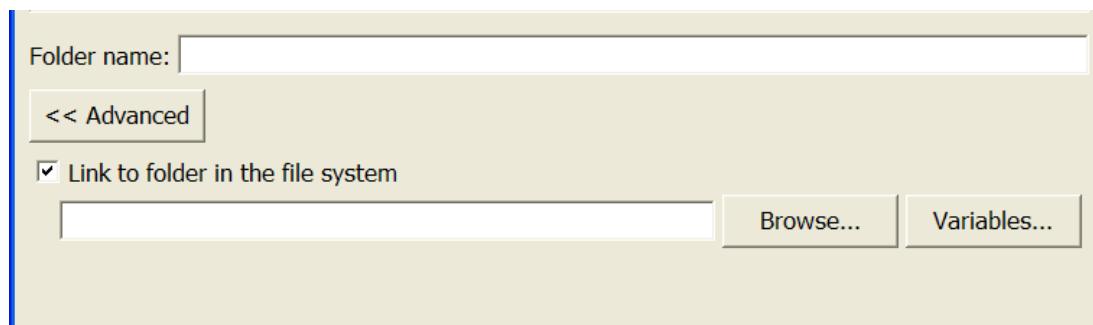
Once you have selected a parent node, you can type the name of a new node.

This node will be created as a child of your selected parent node.

It will be considered as the parent node of your new Database Connection.

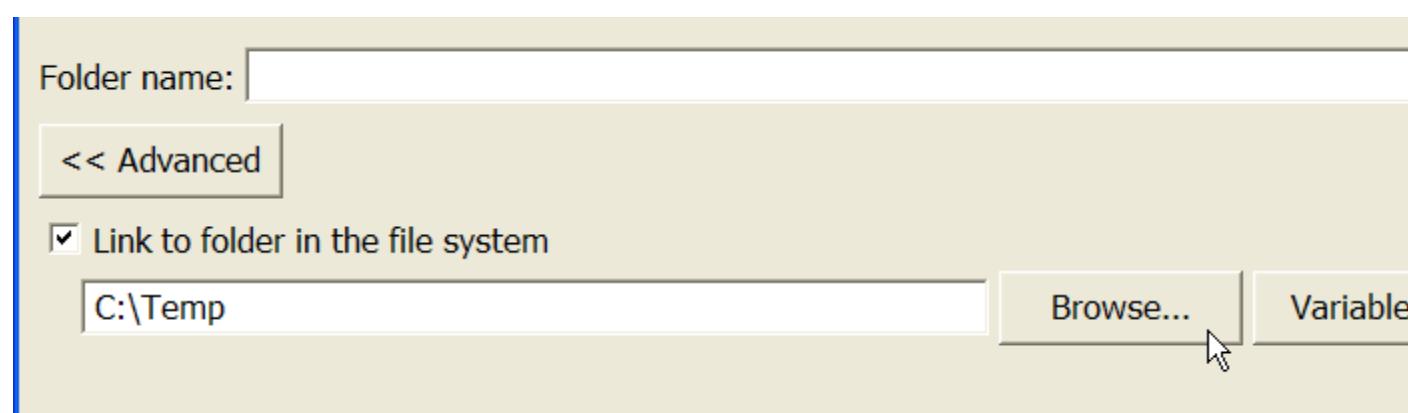
This value is optional.

2.1.3. Advanced

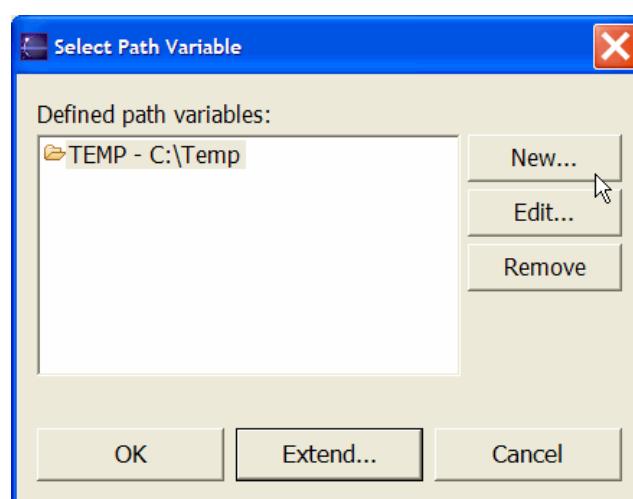


The **Advanced** button lets you create a link to an existing folder in the file system.

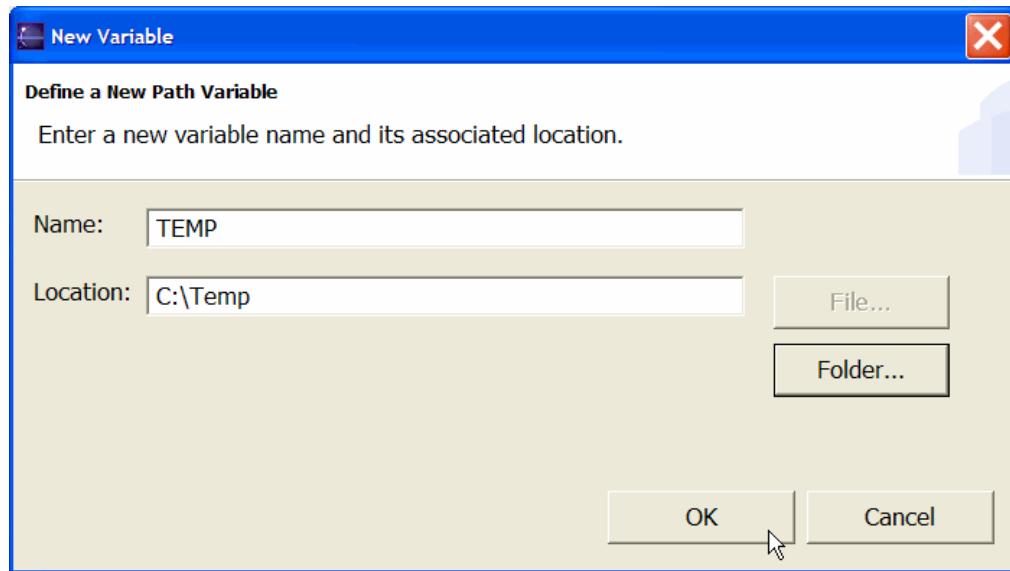
The **Browse** button opens a folder selector.



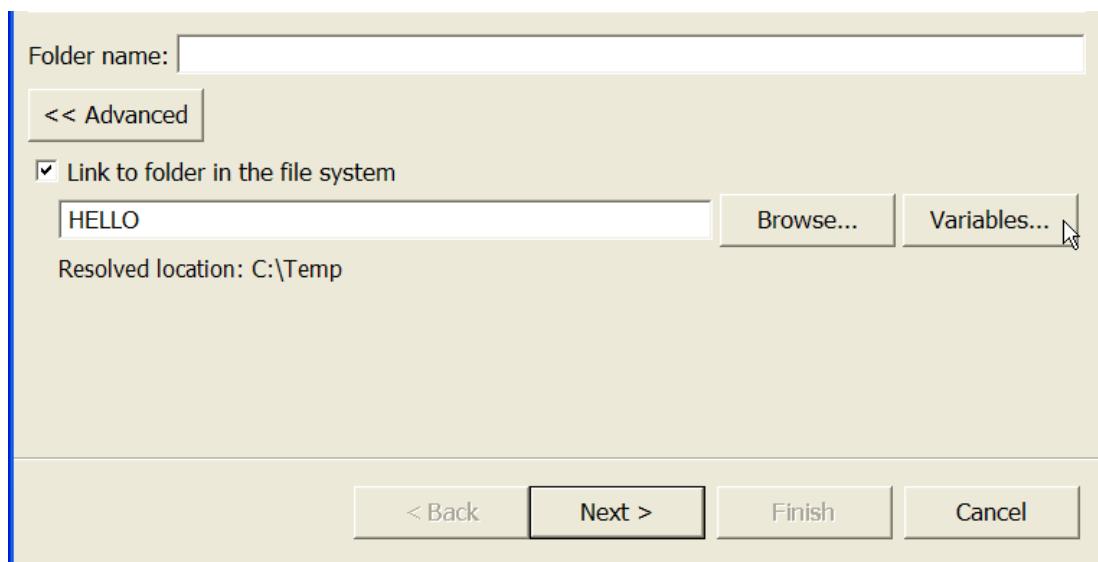
The **Variables** button opens a variable selector.



With this dialog box you can select or create a new Path variable.

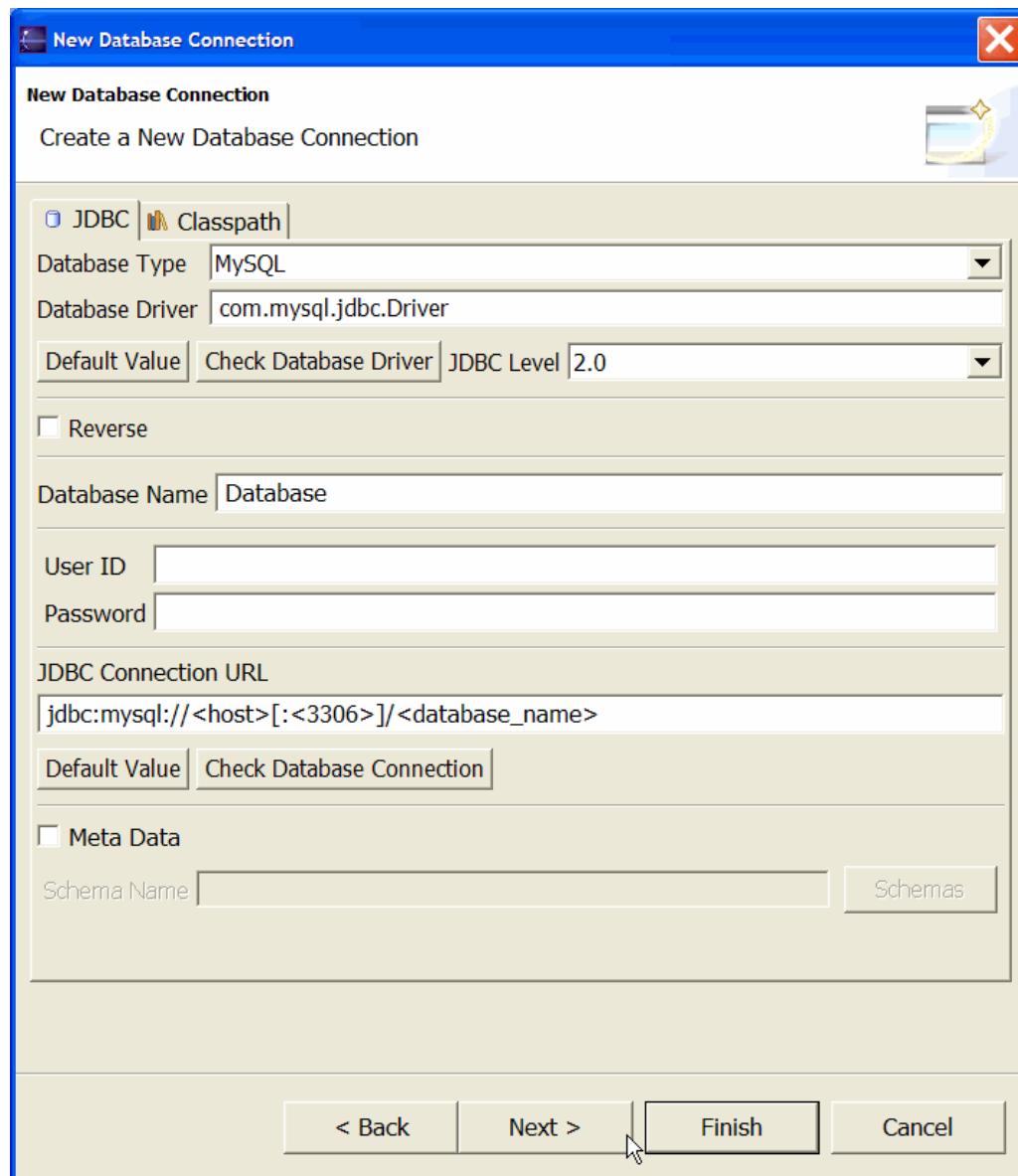


Here is the result in your Database Folder page:



This described mechanism is the standard Eclipse way of working when you create a Folder.

2.2. New Database Connection



The Database Connection page has the same fields as the [Database Connection wizard page](#) in the Database Explorer.

However the [Reverse](#) check box is inherited from the [Connection Global Preferences](#).

It means that a Database Reverse is optionally performed when you create a Database Connection.

2.3. New Database Schema

The Database Schema page has the same fields as the [Database Schema wizard page](#) in the Database Explorer.

2.4. New Database Schema DTD

The Database Schema DTD page has the same fields as the [Database Schema DTD wizard page](#) in the Database Explorer.

2.5. Finish

Once you have defined your connection and pressed the **Finish** button, your Database Connection is created.

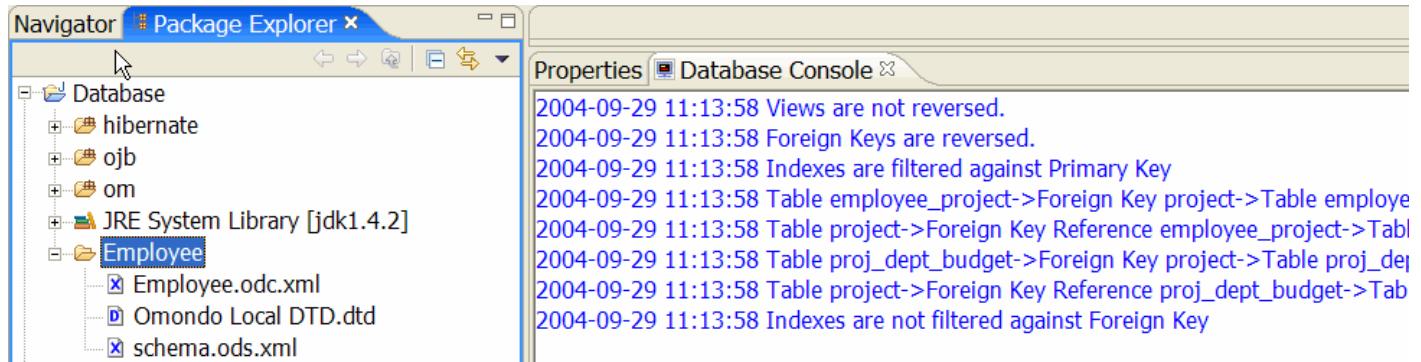
Three files are defined in your workspace :

- A [Database Connection](#) file
- A [Database Schema](#) file
- A [Database Schema DTD](#) file

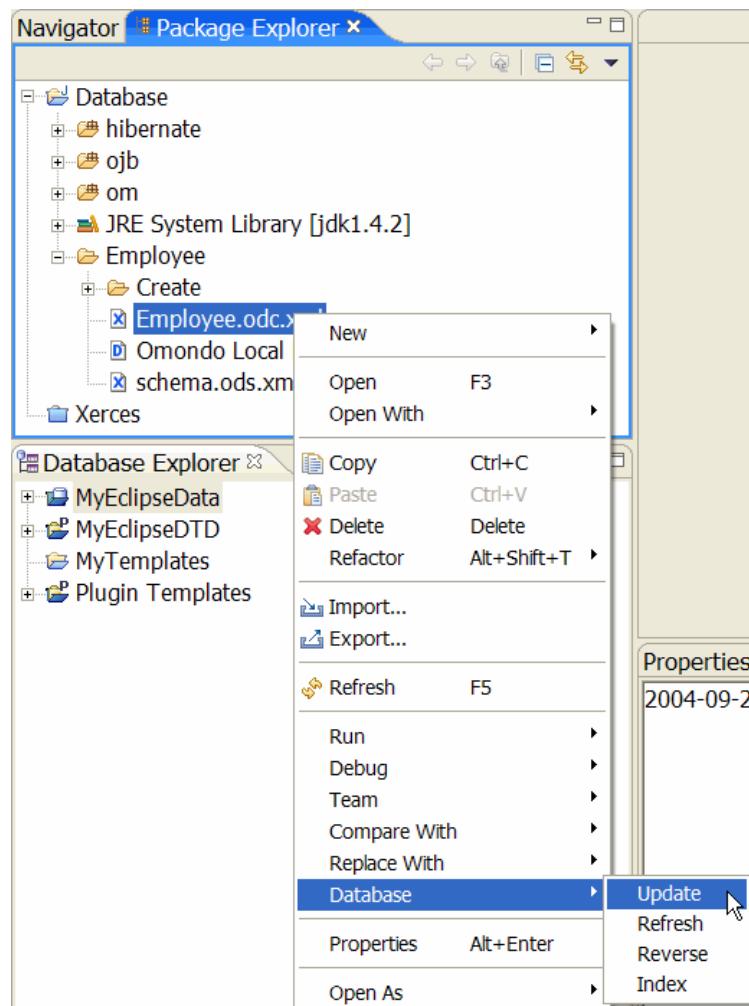
A folder is optionally defined.

If you requested a **Reverse**, a connection will be established to your database and a reverse will be performed.

A report is printed in the **DatabaseConsole**.



3. Database Connection Update



When you want to modify the parameters of an existing Database Connection, select any of its related files, right-click and select

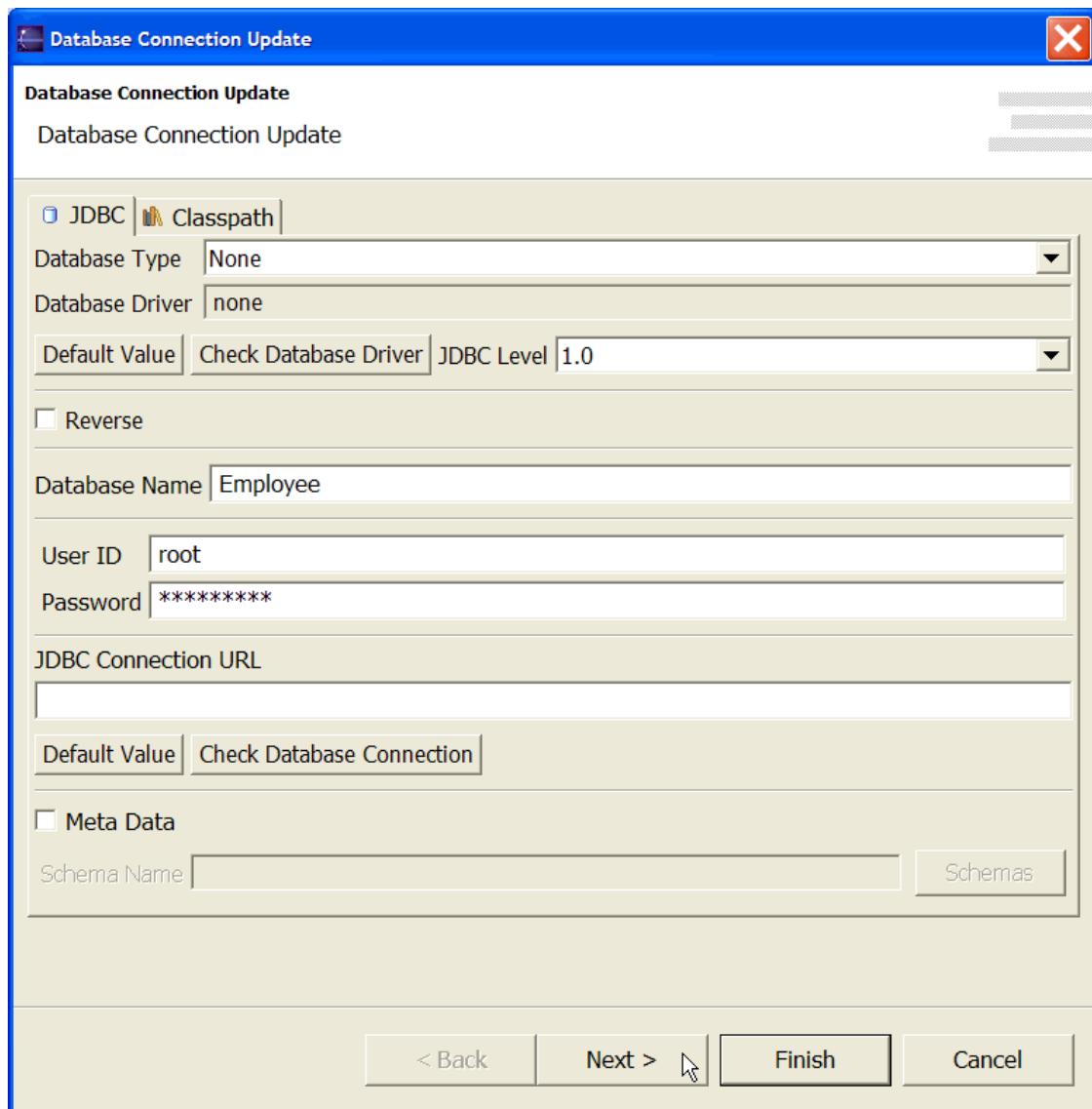
Database->Update

In the [Introduction](#) of this chapter you learned that a selection is contextual.

In the [Overview](#) of this documentation, you saw the UML Class Diagram of a Database Connection. Each file belonging to a Database Connection could be the starting point to open its Database Connection.

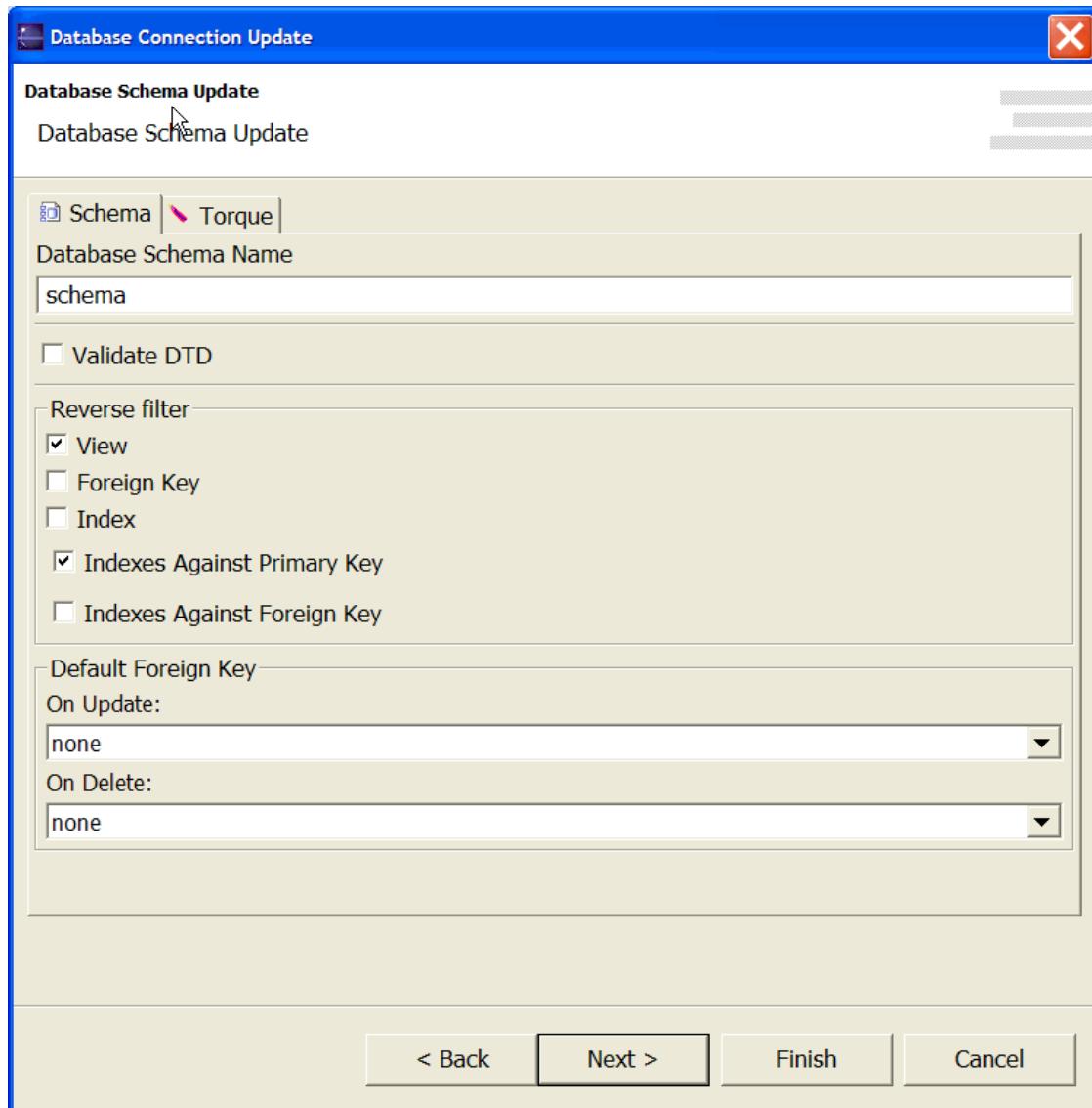
Right-Clicking on an SQL file or a Schema DTD will let you select the **Database->Update** command. This mechanism is managed by an [Index](#).

3.1. Database Connection Update



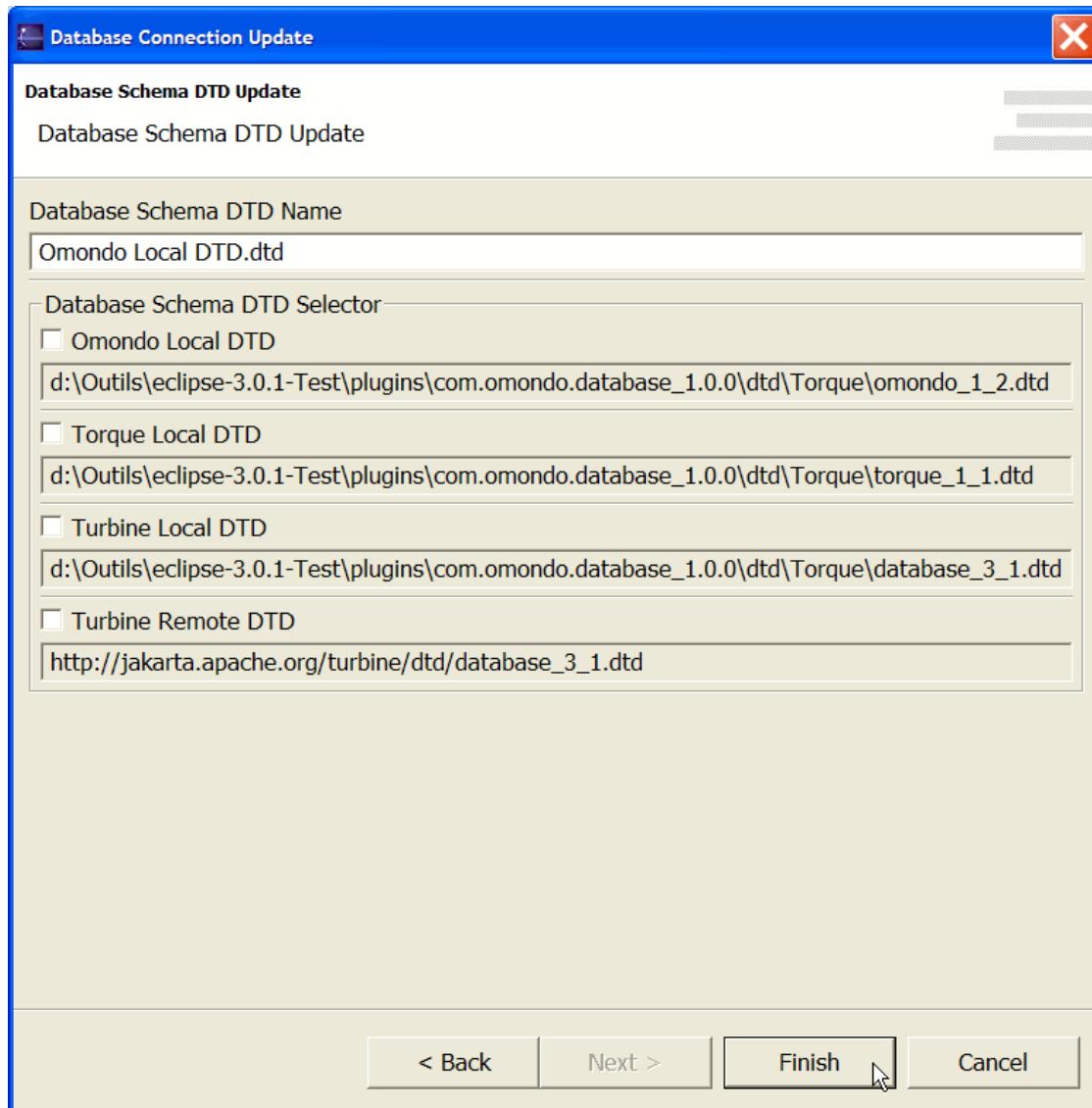
The Database Connection Update page has the same fields as the [Database Connection wizard page](#) in the Database Explorer.

3.2. Database Schema Update



The Database Schema Update page has the same fields as the [Database Schema wizard page](#) in the Database Explorer.

3.3. Database Schema DTD Update



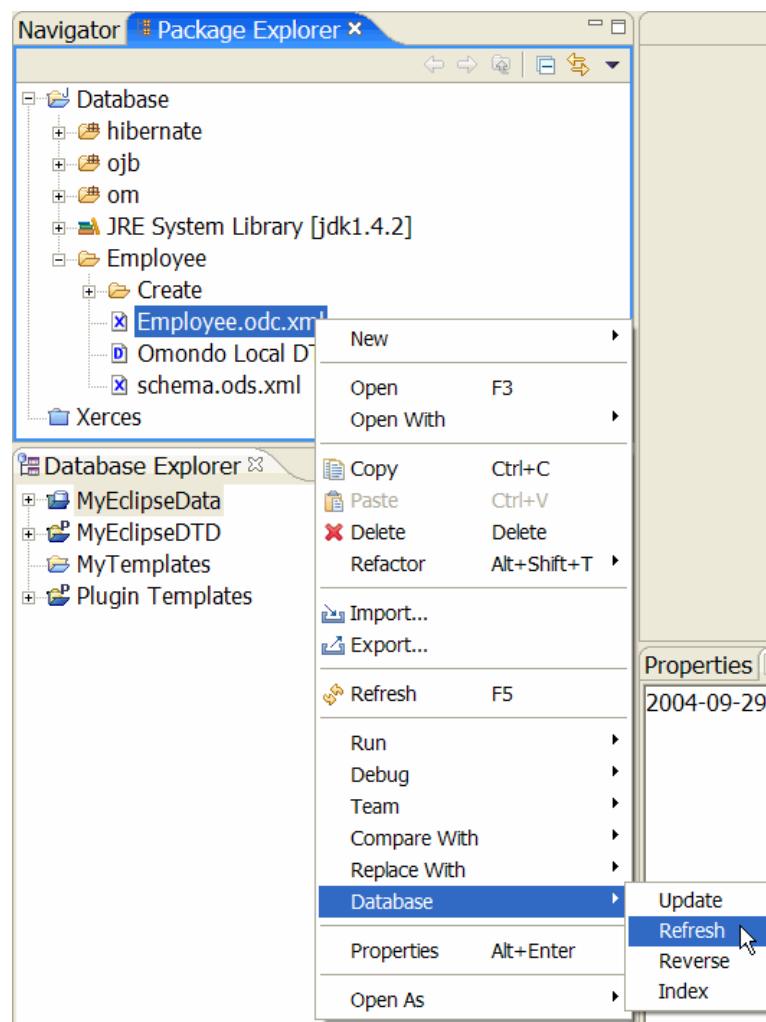
The Database Schema DTD Update page has the same fields as the [Database Schema DTD wizard wage](#) in the Database Explorer.

In update mode the wizard doesn't know which kind of DTD you previously selected.

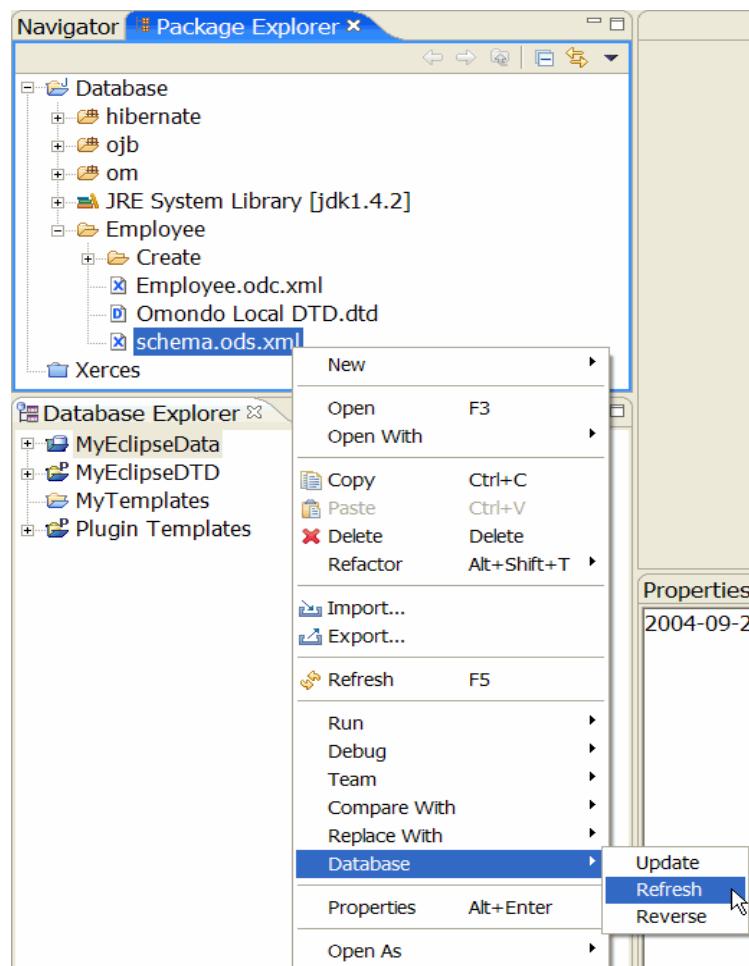
4. Refresh

Select a Database Object in the Package Explorer View or the Navigator View then right-click and select :

Database->Refresh



If a Database Connection file is refreshed, then it will be parsed.



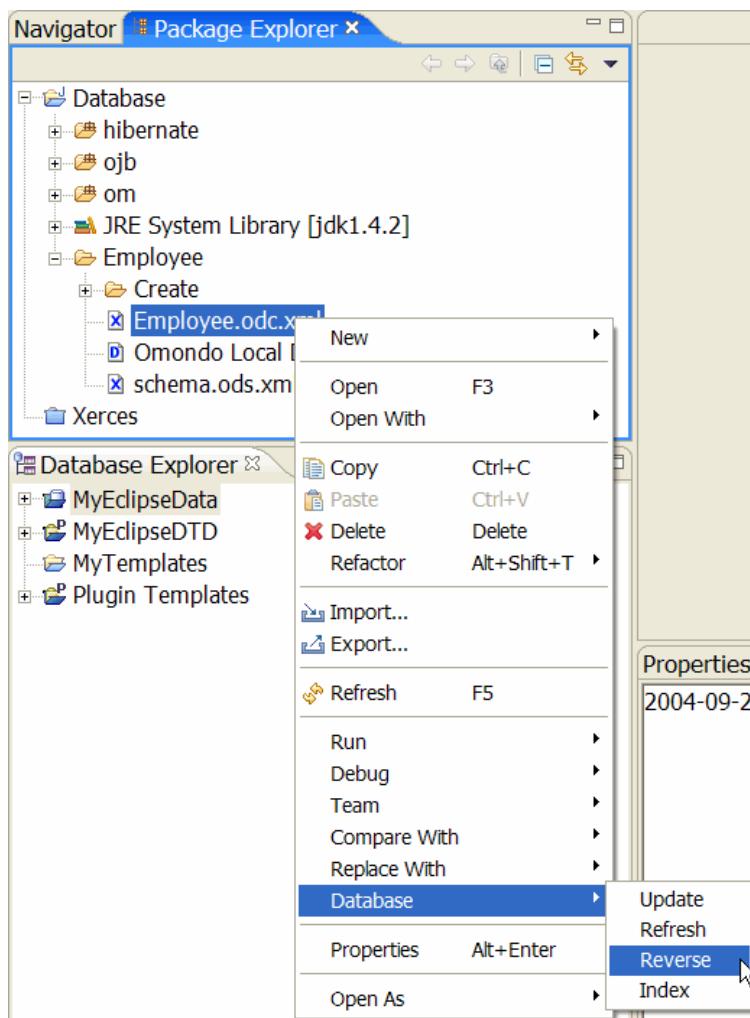
If a Database Schema or a Database Schema DTD file is refreshed, then the Database Schema file will be parsed.

As a general rule this command reads the selected Database Object.

5. Reverse

Select a Database Object in the Package Explorer View or the Navigator View then right-click and select :

Database->Reverse



The Database Meta Data will be reversed and a merge process will be performed with the current contents of your Database Schema.

A report will be displayed in the [DatabaseConsole](#).

Properties Database Console

```

2004-09-29 14:35:20 Table employee_project->Foreign Key project->Table employee_project->Column PROJ_ID->Ta
2004-09-29 14:35:20 Table project->Foreign Key Reference employee_project->Table employee_project->Column PRC
2004-09-29 14:35:20 Table proj_dept_budget->Foreign Key project->Table proj_dept_budget->Column PROJ_ID->Ta
2004-09-29 14:35:20 Table project->Foreign Key Reference proj_dept_budget->Table proj_dept_budget->Column PRI
2004-09-29 14:35:20 Views are not reversed.
2004-09-29 14:35:20 Foreign Keys are reversed.
2004-09-29 14:35:20 Indexes are filtered against Primary Key
2004-09-29 14:35:20 Table employee_project->Foreign Key project->Table employee_project->Column PROJ_ID->Ta
2004-09-29 14:35:20 Table project->Foreign Key Reference employee_project->Table employee_project->Column PRC
2004-09-29 14:35:20 Table proj_dept_budget->Foreign Key project->Table proj_dept_budget->Column PROJ_ID->Ta
2004-09-29 14:35:20 Table project->Foreign Key Reference proj_dept_budget->Table proj_dept_budget->Column PRI
2004-09-29 14:35:20 Indexes are not filtered against Foreign Key

```

Database Create SQL Script

1. Introduction
2. Database Create SQL Script
 1. Database Create SQL Script Page
 1. Script Name
 2. Database Name
 2. Finish
3. Templates
4. Additional Resources

1. Introduction

The purpose of this chapter is to show how to generate a Create SQL Script.

When forwarded to your Database, this script will be able to initialize a Database.

2. Database Create SQL Script

To start the Database Create SQL Script wizard, select :

File->New->Other->Database->Database Create SQL Script

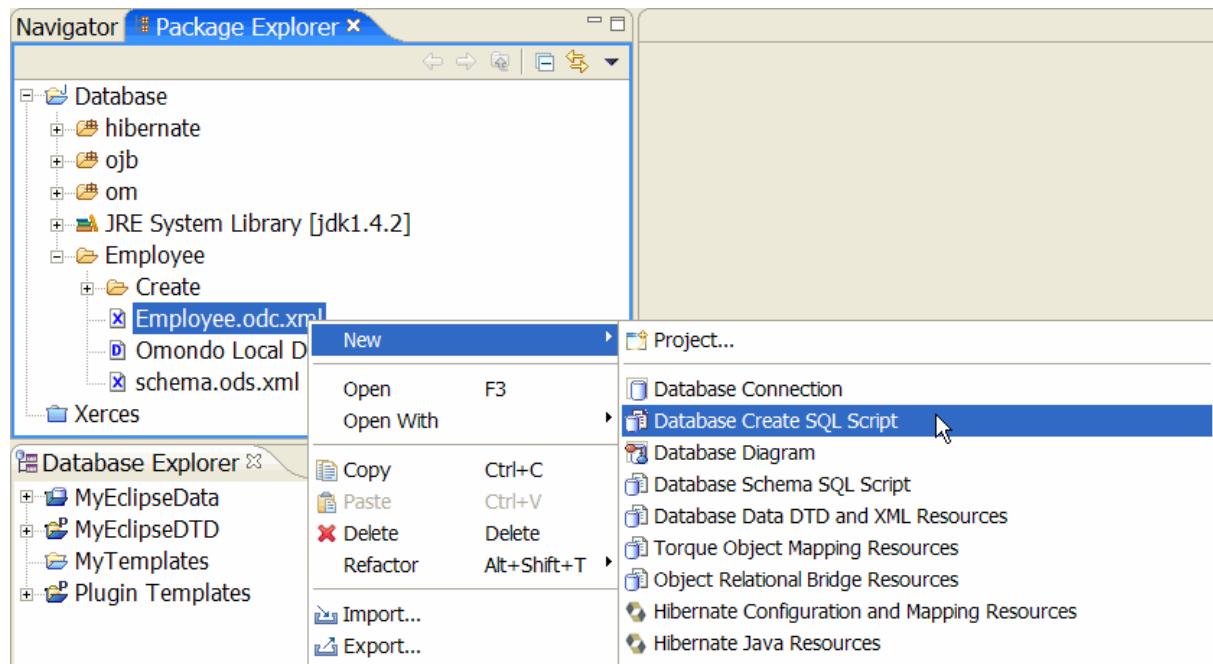
or

Select an object in the Package Explorer View or the Navigator View then right-click and select :

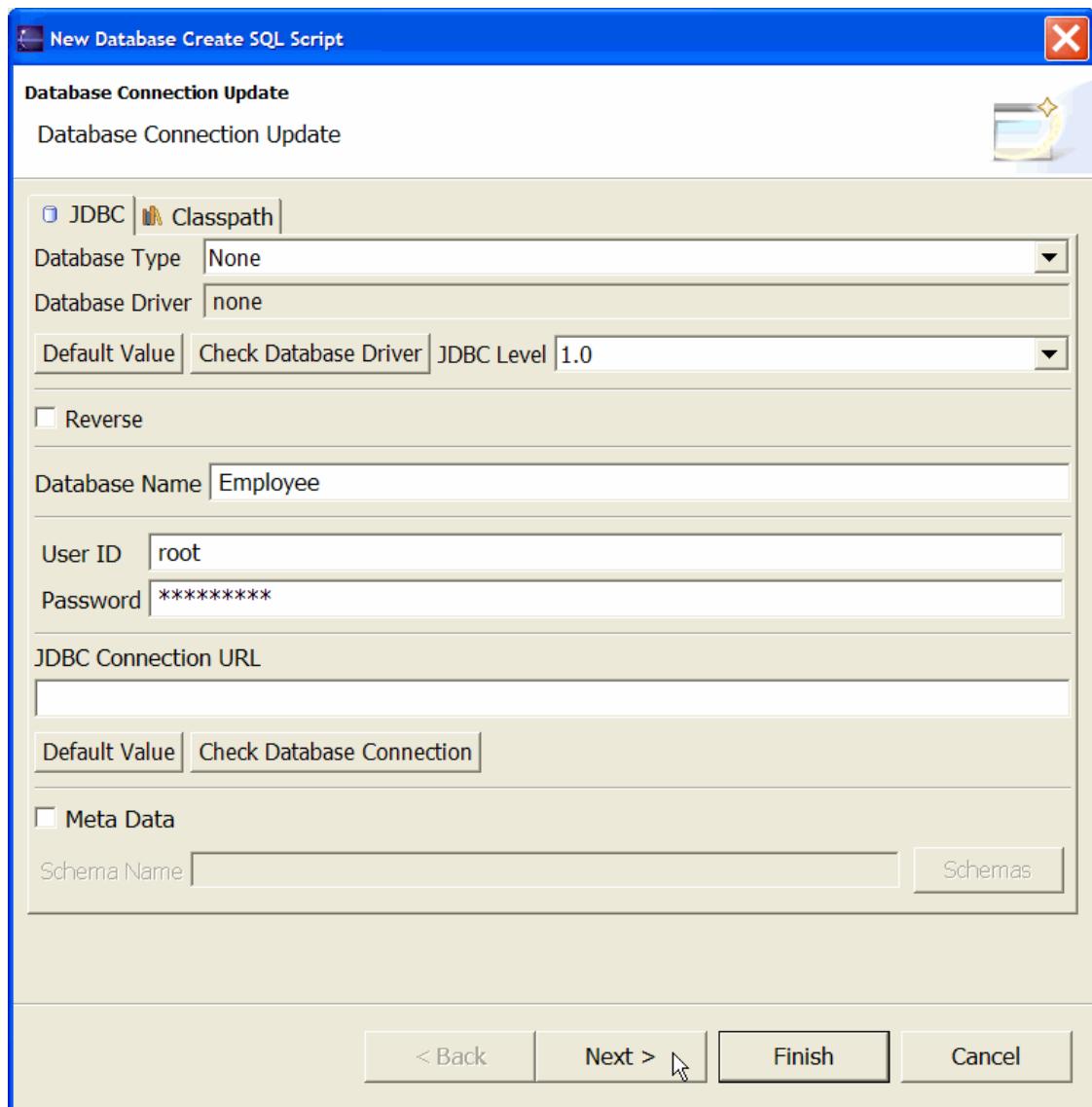
New->Other->Database->Database Create SQL Script

The selection is contextual.

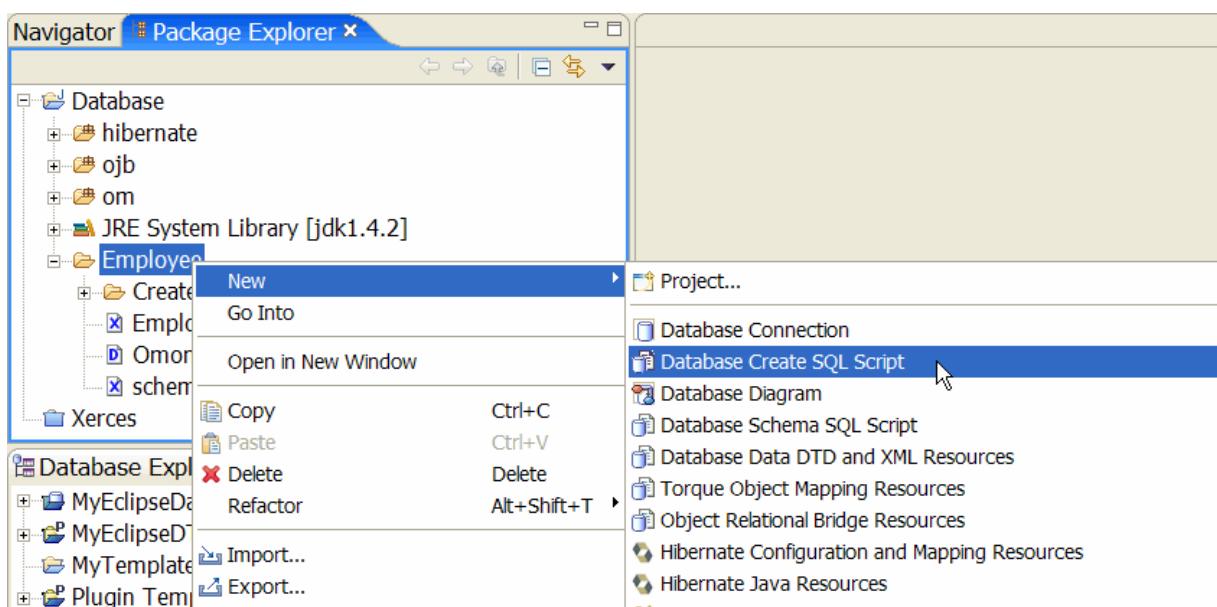
This means that if you select an existing Database Object:



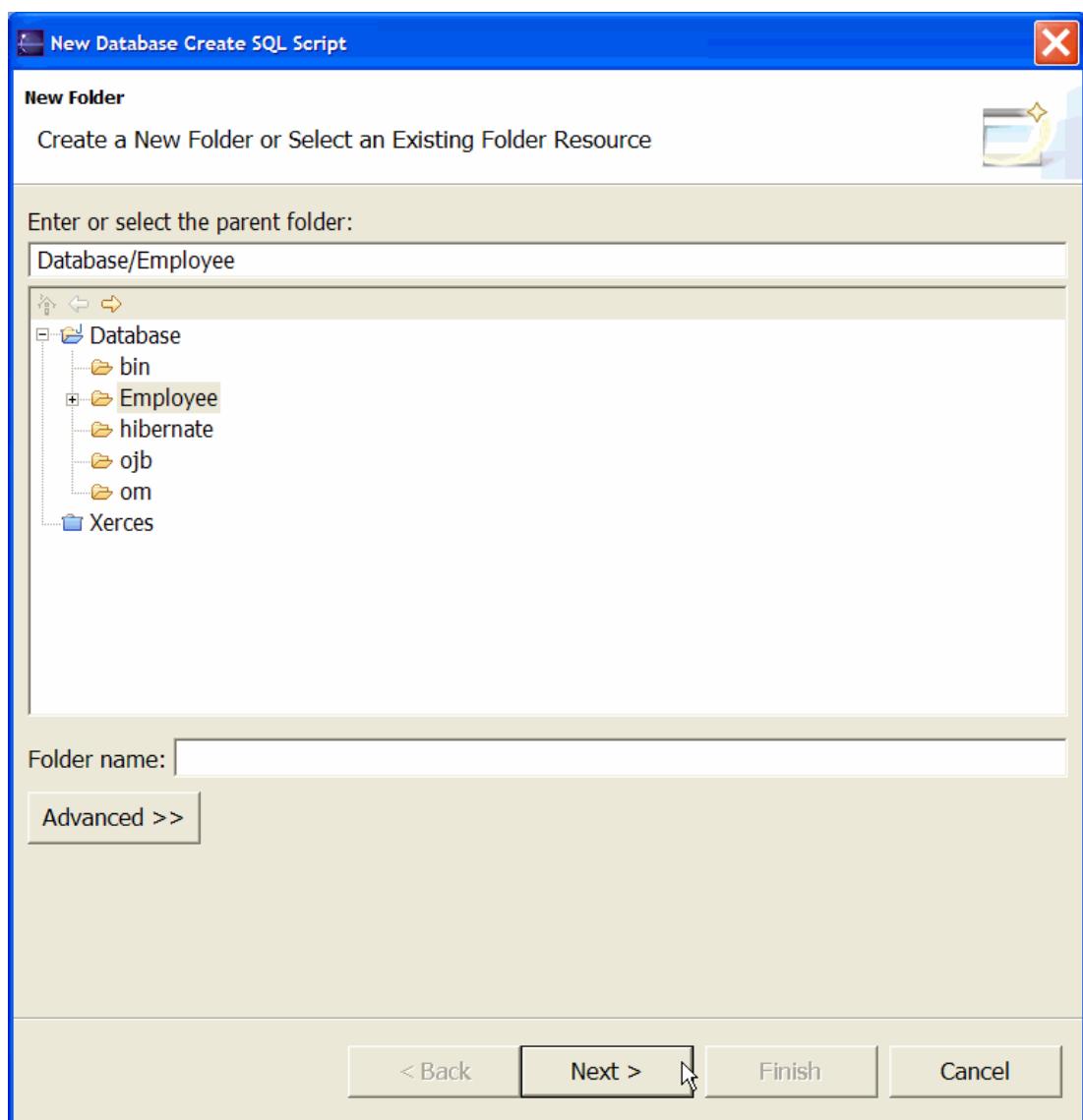
The wizard will open the parent Database Connection in update mode.



Otherwise:



A new Database Connection wizard will be opened starting with a **New Folder** selector.



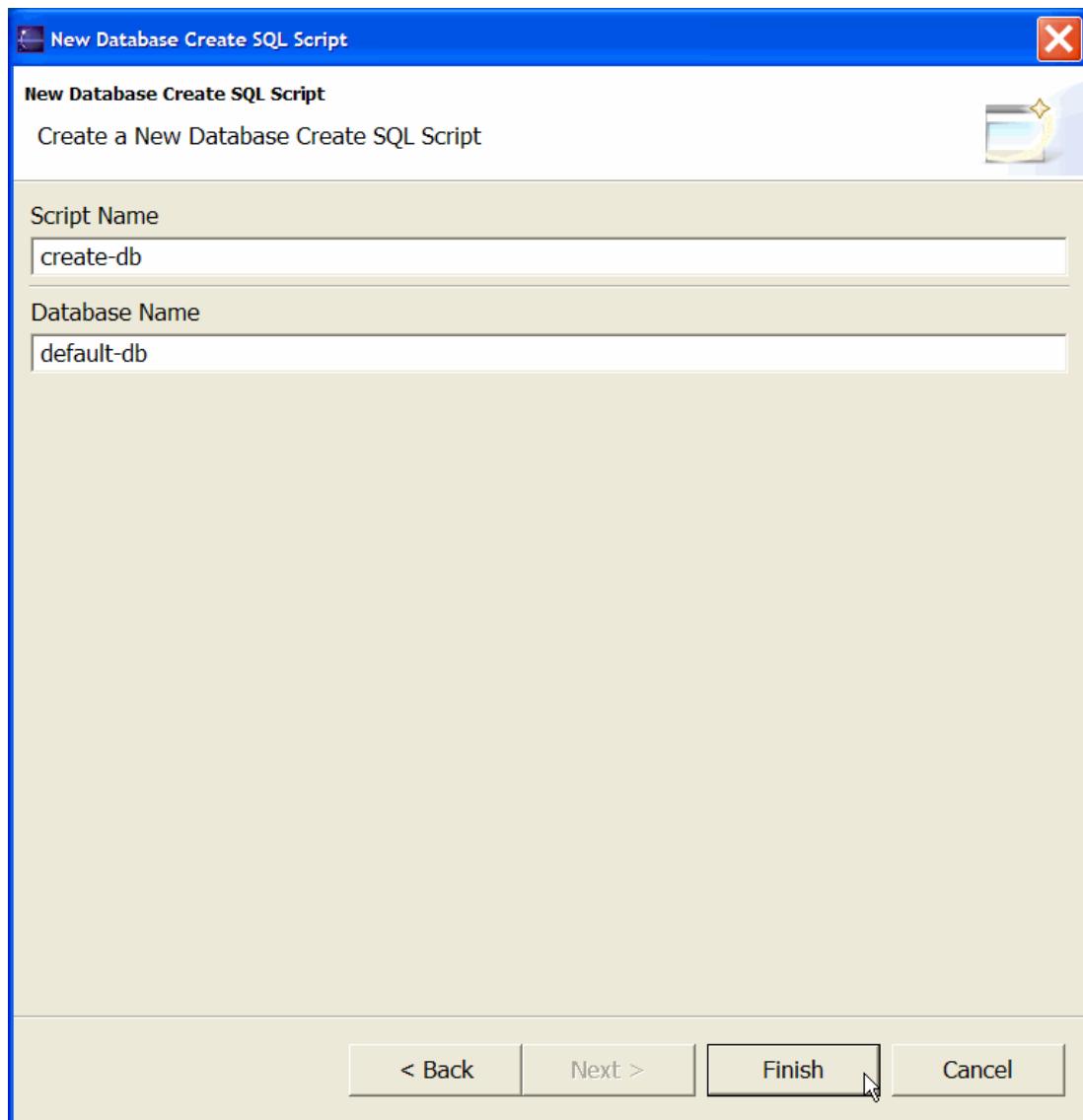
In Database Connection Update mode this wizard has four pages :

- Database Connection Update
- Database Schema Update
- Database Schema DTD Update
- New Database Create SQL Script

In Database Connection Create mode this wizard has five pages :

- New Folder
- New Database Connection
- New Database Schema
- New Database Schema DTD
- New Database Create SQL Script

2.1. Database Create SQL Script Page



2.1.1. Script Name

Script Name is the name of your new SQL script.

Each resource should have a unique name in the targeted selected workspace folder or project root.

This data is not related to any Meta Data informations.

This name will be the file name of the generated SQL script file.

A Database Create Script is an SQL file, its file extension is :

- .sql

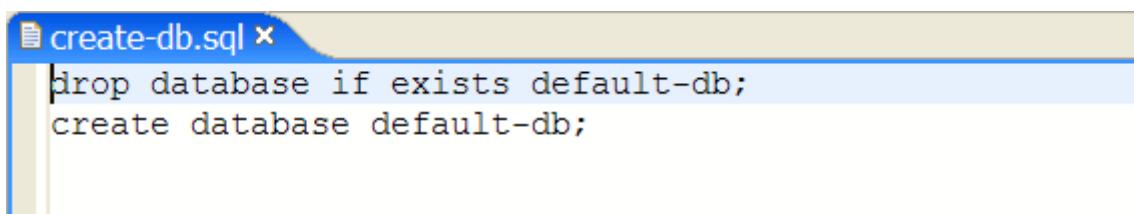
This file is referenced in your current [Database Connection](#).

2.1.2. Database Name

Database Name is the name of your database.

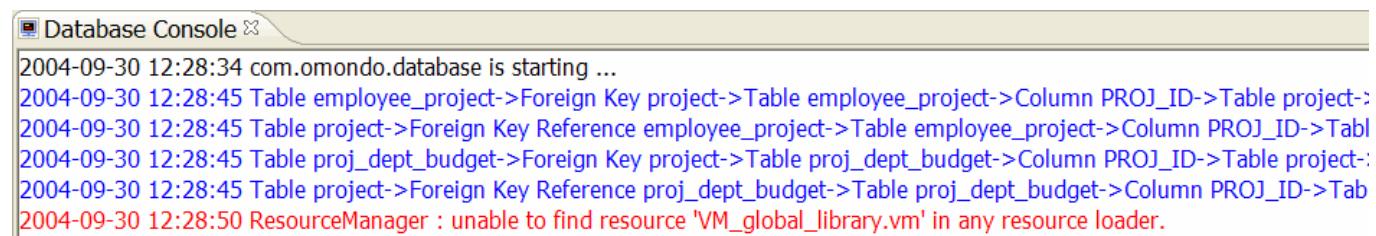
2.2. Finish

The script generation of the new Database Create SQL Script will be processed and created.



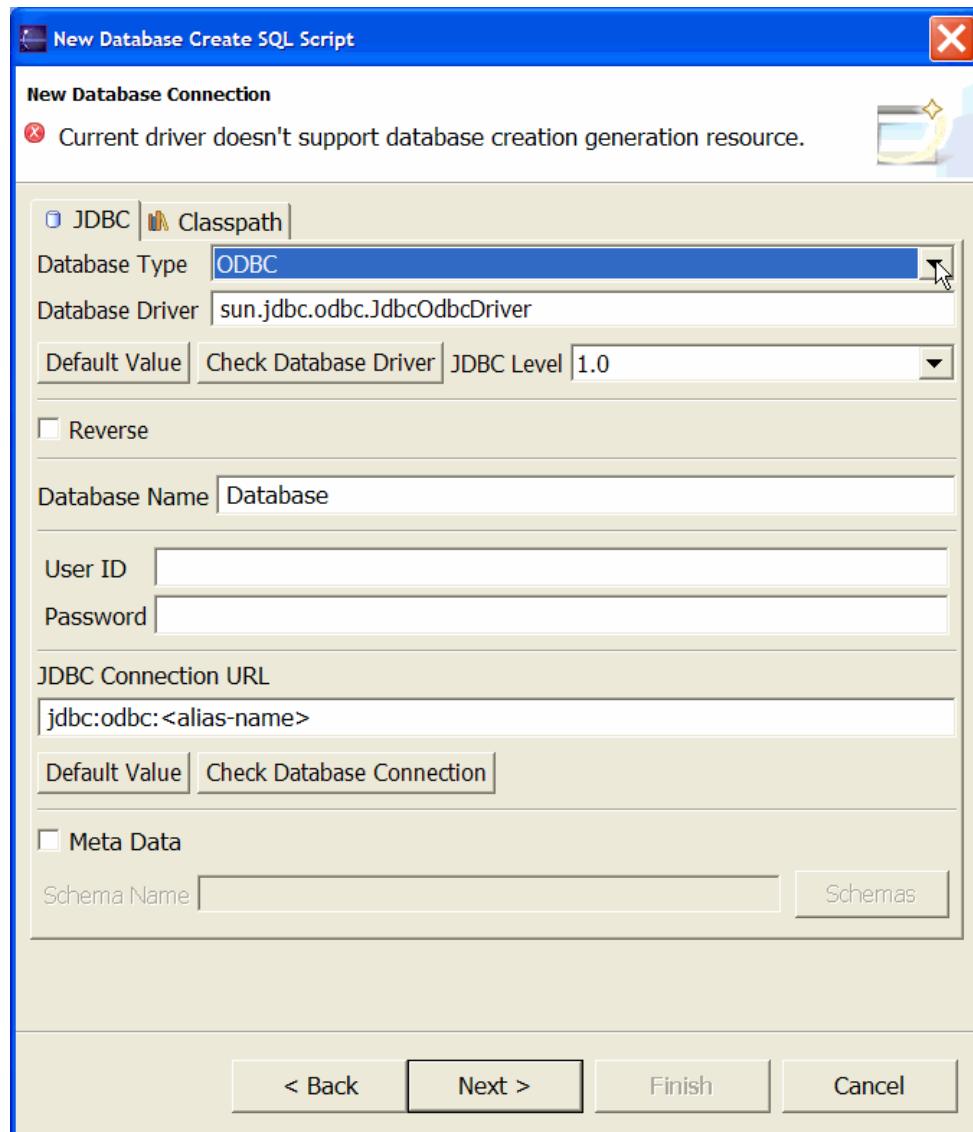
```
create-db.sql
drop database if exists default-db;
create database default-db;
```

Detailed information is available in the [DatabaseConsole](#) window.



```
Database Console
2004-09-30 12:28:34 com.omondo.database is starting ...
2004-09-30 12:28:45 Table employee_project->Foreign Key project->Table employee_project->Column PROJ_ID->Table project-
2004-09-30 12:28:45 Table project->Foreign Key Reference employee_project->Table employee_project->Column PROJ_ID->Tabl
2004-09-30 12:28:45 Table proj_dept_budget->Foreign Key project->Table proj_dept_budget->Column PROJ_ID->Table project-
2004-09-30 12:28:45 Table project->Foreign Key Reference proj_dept_budget->Table proj_dept_budget->Column PROJ_ID->Tab
2004-09-30 12:28:50 ResourceManager : unable to find resource 'VM_global_library.vm' in any resource loader.
```

Some databases don't support the drop and create statements.
A message is displayed in the Database Connection page.

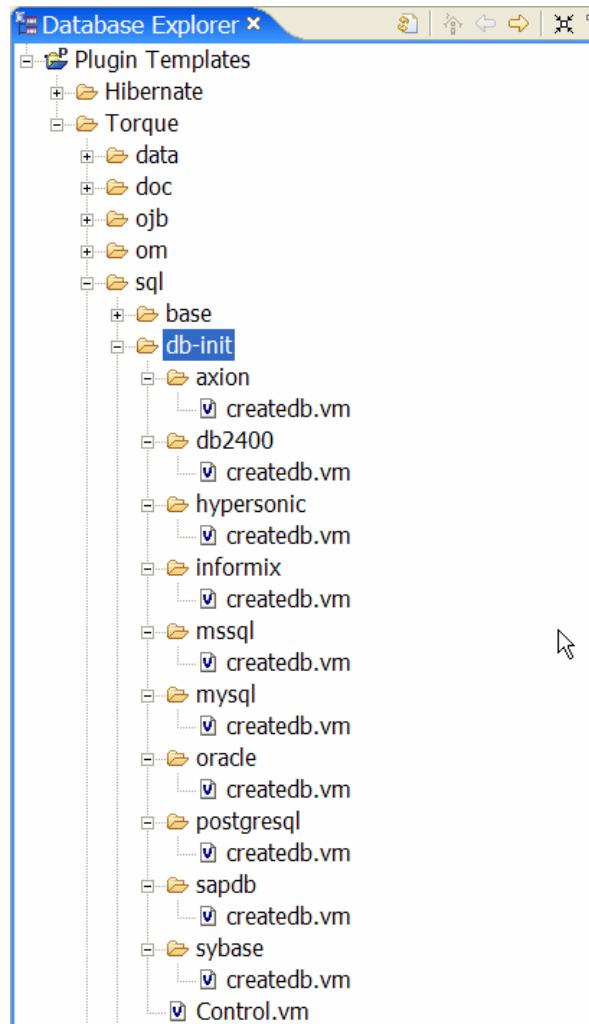


3. Templates

This wizard is [Template](#) based.

Below your [Plugin Templates](#) or [MyTemplates](#) you should see the following directory:

Torque/sql/db-init



These templates are [Velocity](#) based and use the [Torque Model API](#).

4. Additional Resources

Here are several links provided for further Apache Velocity related information.

- [Apache Velocity Project](#)
- [Velocity Mailing Lists](#)

Database Diagram

1. [Introduction](#)
2. [Database Diagram](#)
 1. [Database Diagram Page](#)
 1. [Database Diagram Name](#)
 2. [Finish](#)

1. Introduction

The purpose of this chapter is to show how to generate a Database Diagram.

According to the [Database Connection UML Diagram](#) you can create as many diagrams as you like.

2. Database Diagram

To start the Database Diagram wizard, select :

File->New->Other->Database->Database Diagram

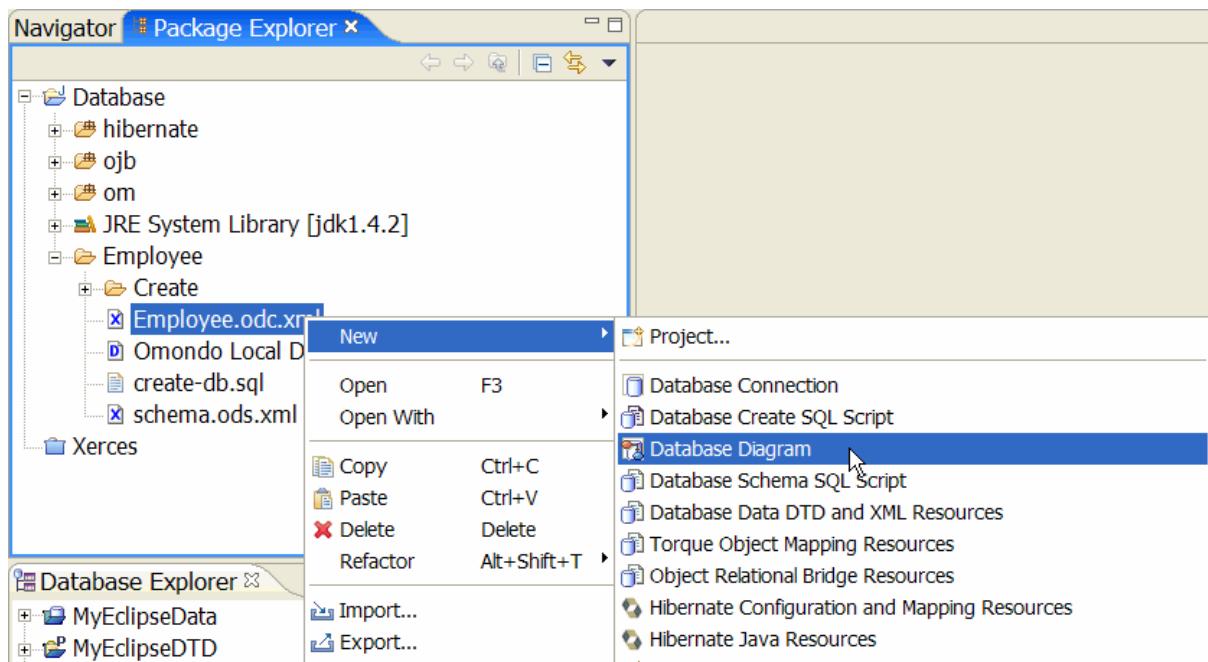
or

Select an object in the Package Explorer View or the Navigator View then right-click and select :

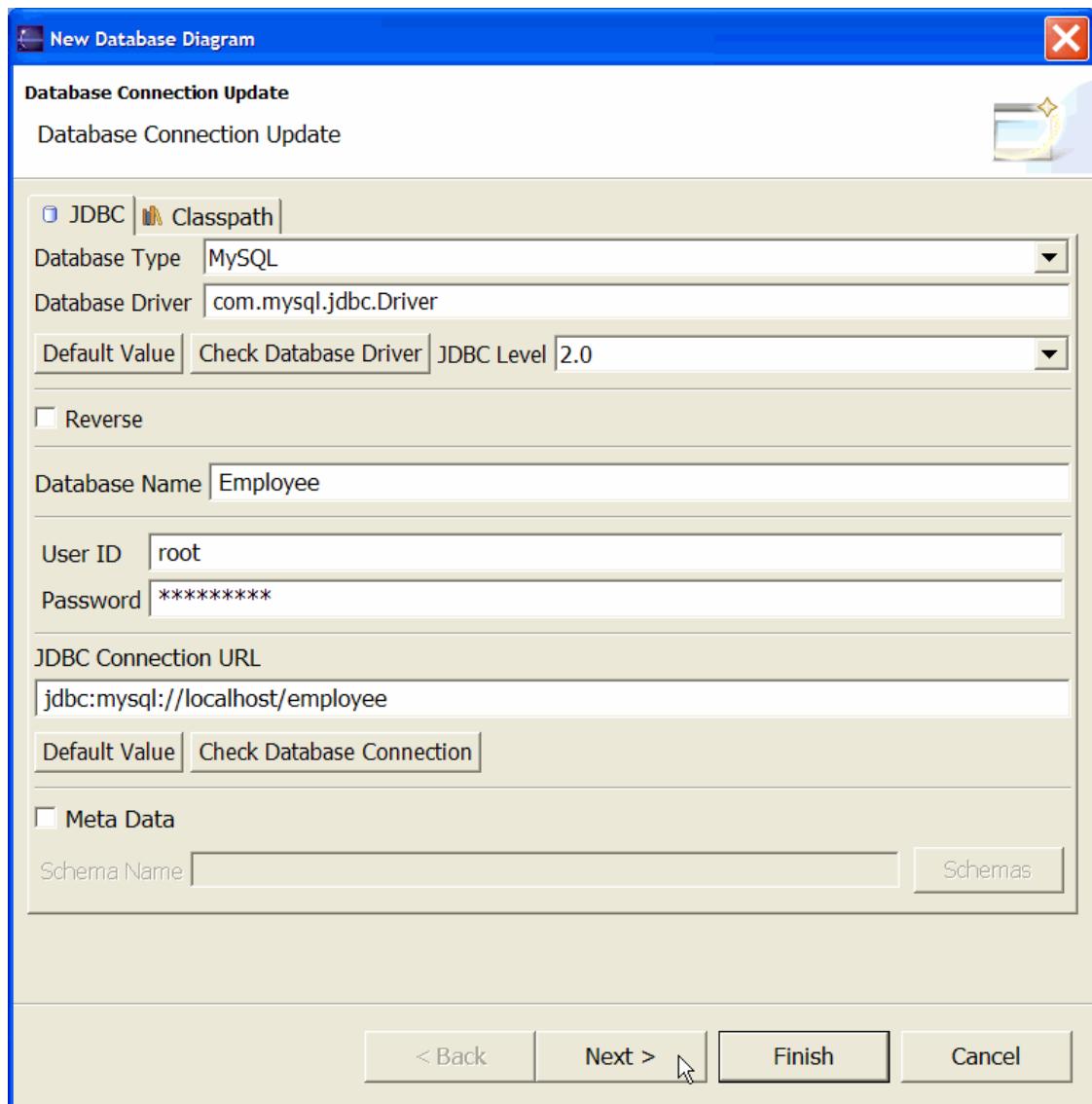
New->Other->Database->Database Diagram

The selection is contextual.

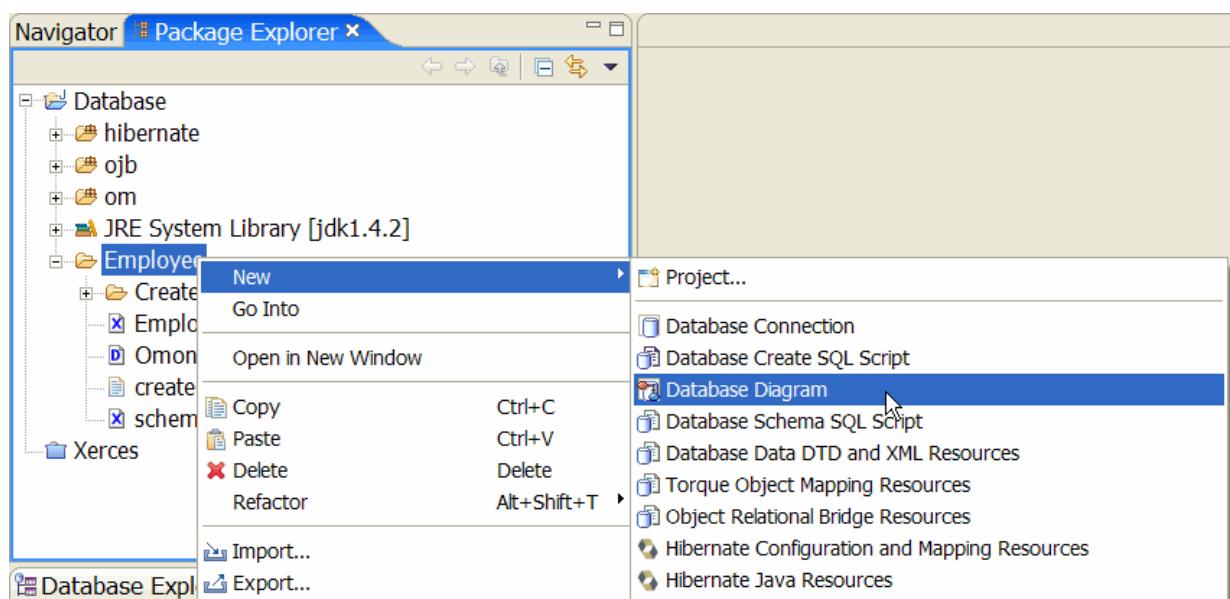
It means that if you select an existing Database Object:



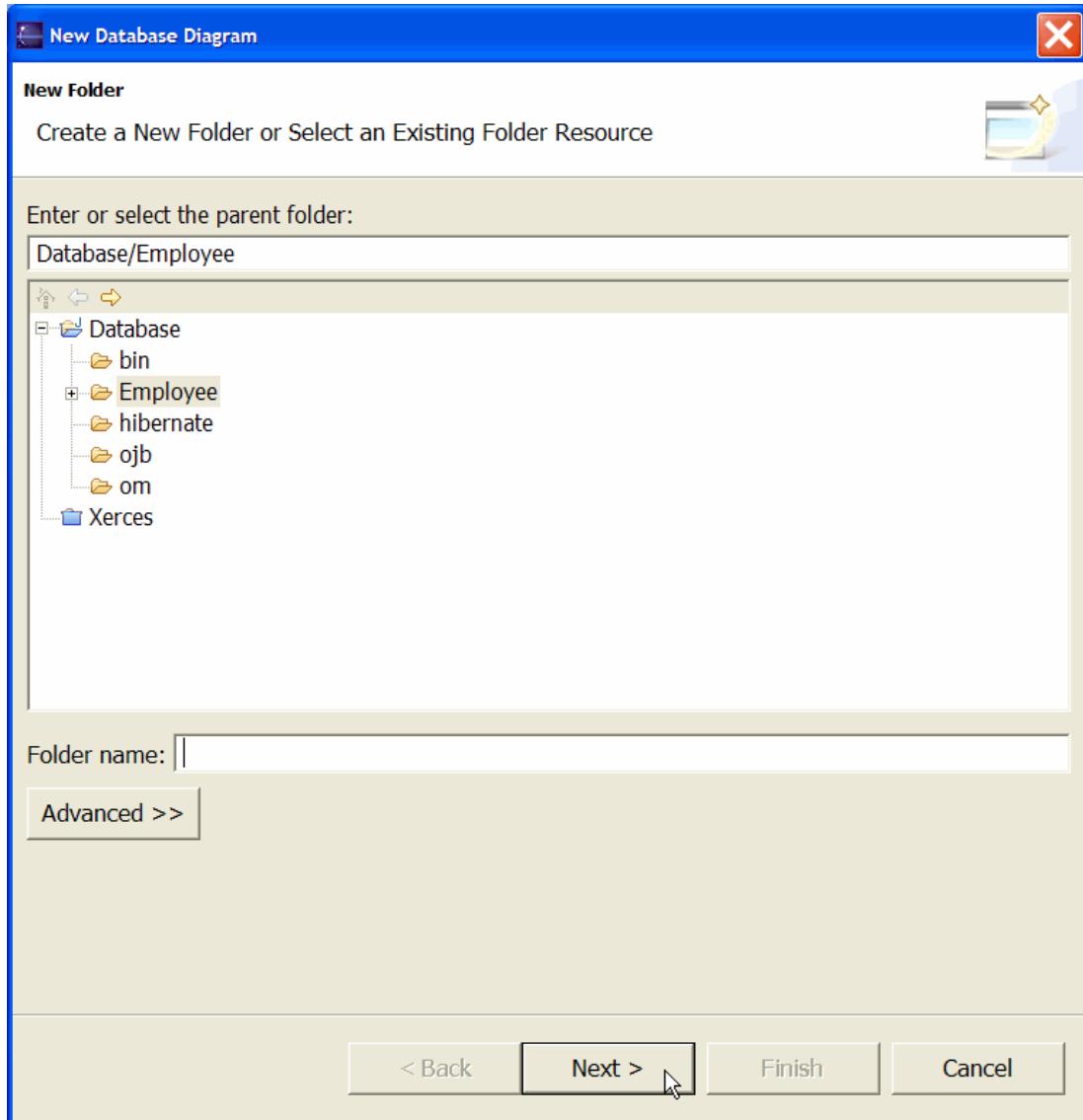
The wizard will open the parent Database Connection in update mode.



Otherwise :



A new Database Connection wizard will be opened starting with a [New Folder](#) selector.



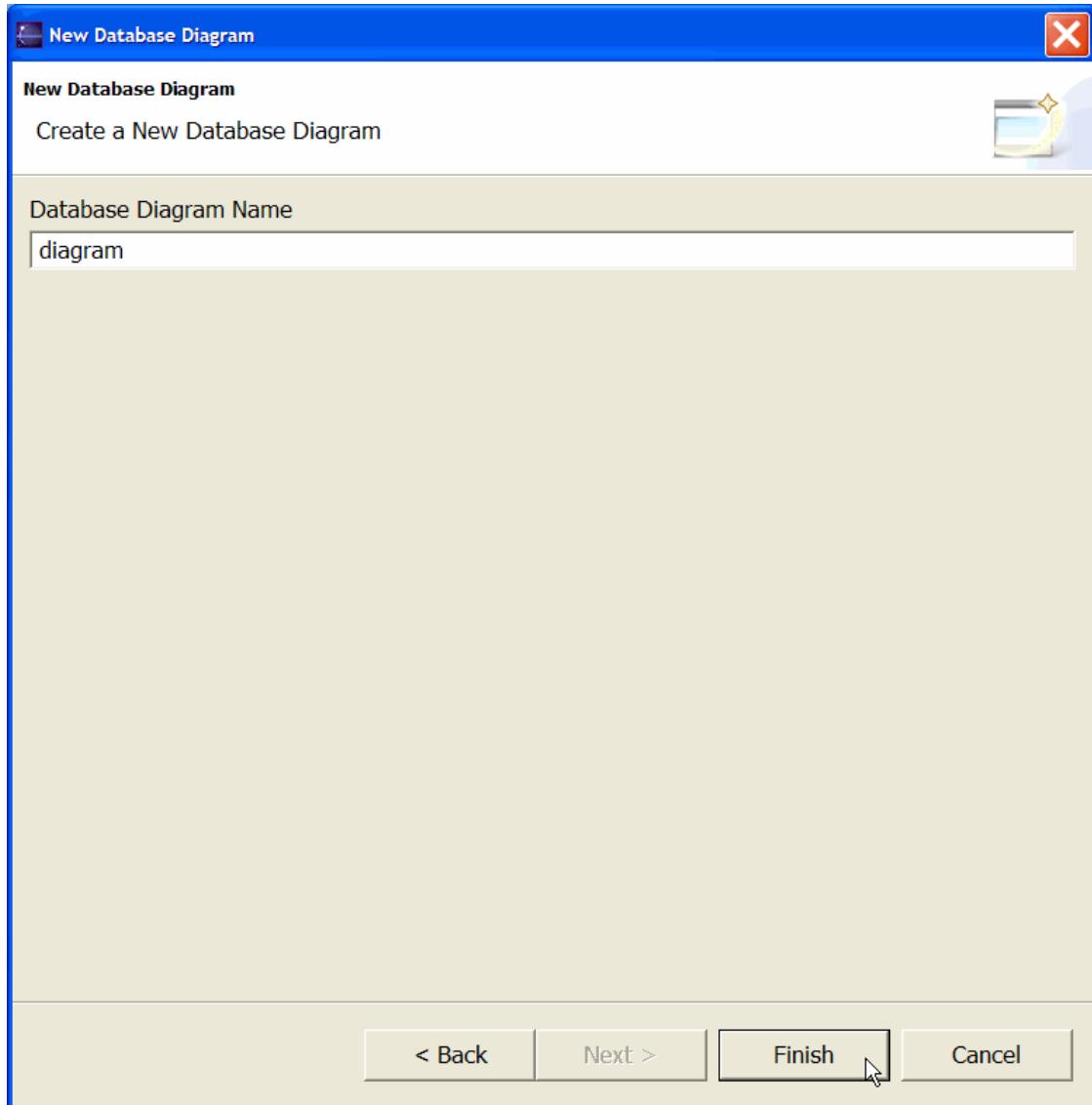
In Database Connection Update mode this wizard has four pages :

- [Database Connection Update](#)
- [Database Schema Update](#)
- [Database Schema DTD Update](#)
- [New Database Diagram](#)

In Database Connection Create mode this wizard has five pages :

- [New Folder](#)
- [New Database Connection](#)
- [New Database Schema](#)
- [New Database Schema DTD](#)
- [New Database Diagram](#)

2.1. Database Diagram Page



2.1.1. Database Diagram Name

Database Diagram Name is the name of your new diagram.

Each resource should have a unique name in the targeted selected workspace folder or project root.

This data is not related to any Meta Data informations.

This name will be the file name of the generated Diagram file.

A Database Diagram is an XML file, its file extension could be :

- .odd.xml
- .odd

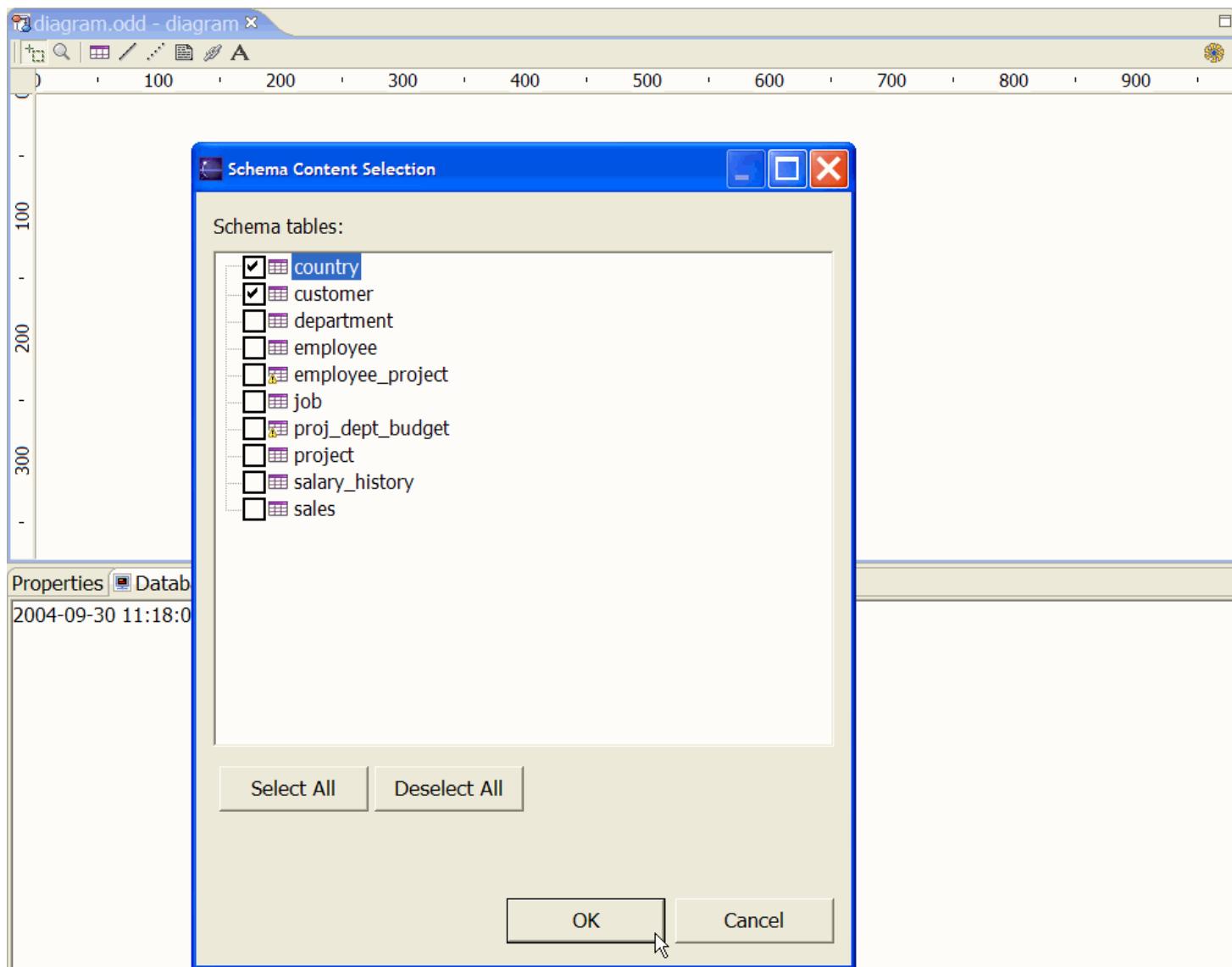
This file extension is managed by the [Save Policy](#) Preference.

This file is referenced in your current [Database Connection](#).

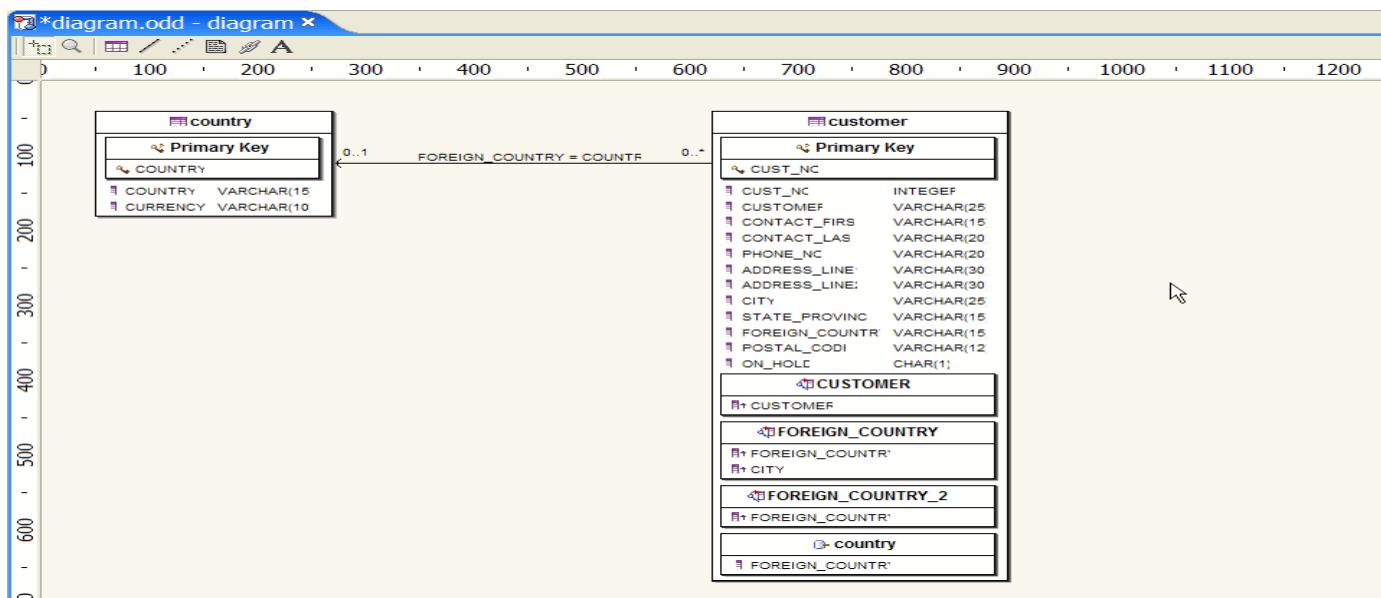
2.2. Finish

The new Database Diagram will be created.

The Diagram will be opened in its editor and a Schema selector will be proposed.



Once you select the Tables you are interested in, the Diagram will display these objects.



Database Schema SQL Script

1. [Introduction](#)
2. [Database Schema SQL Script](#)
 1. [Database Schema SQL Script Page](#)
 1. [Script Name](#)
 2. [Finish](#)
 3. [Templates](#)

1. Introduction

The purpose of this chapter is to show how to generate a Schema SQL Script.

When forwarded to your Database, this script will be able to define the Meta Data of your Database.

2. Database Schema SQL Script

To start the Database Schema SQL Script wizard, select :

File->New->Other->Database->Database Schema SQL Script

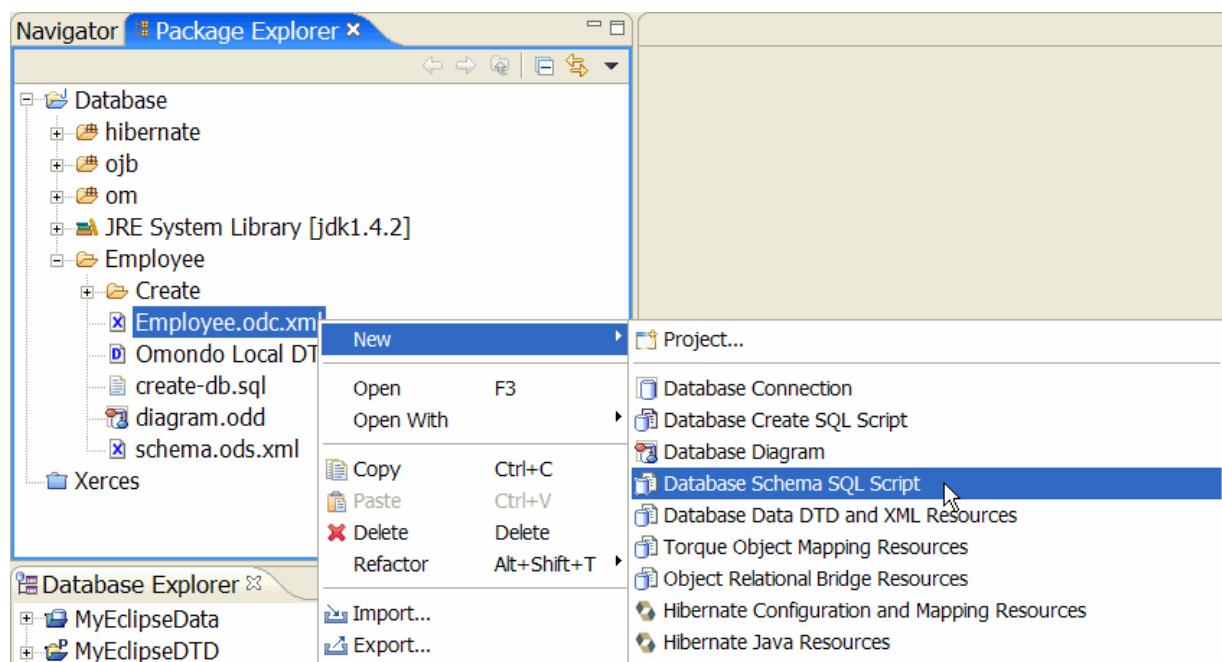
or

Select an object in the Package Explorer View or the Navigator View then right-click and select :

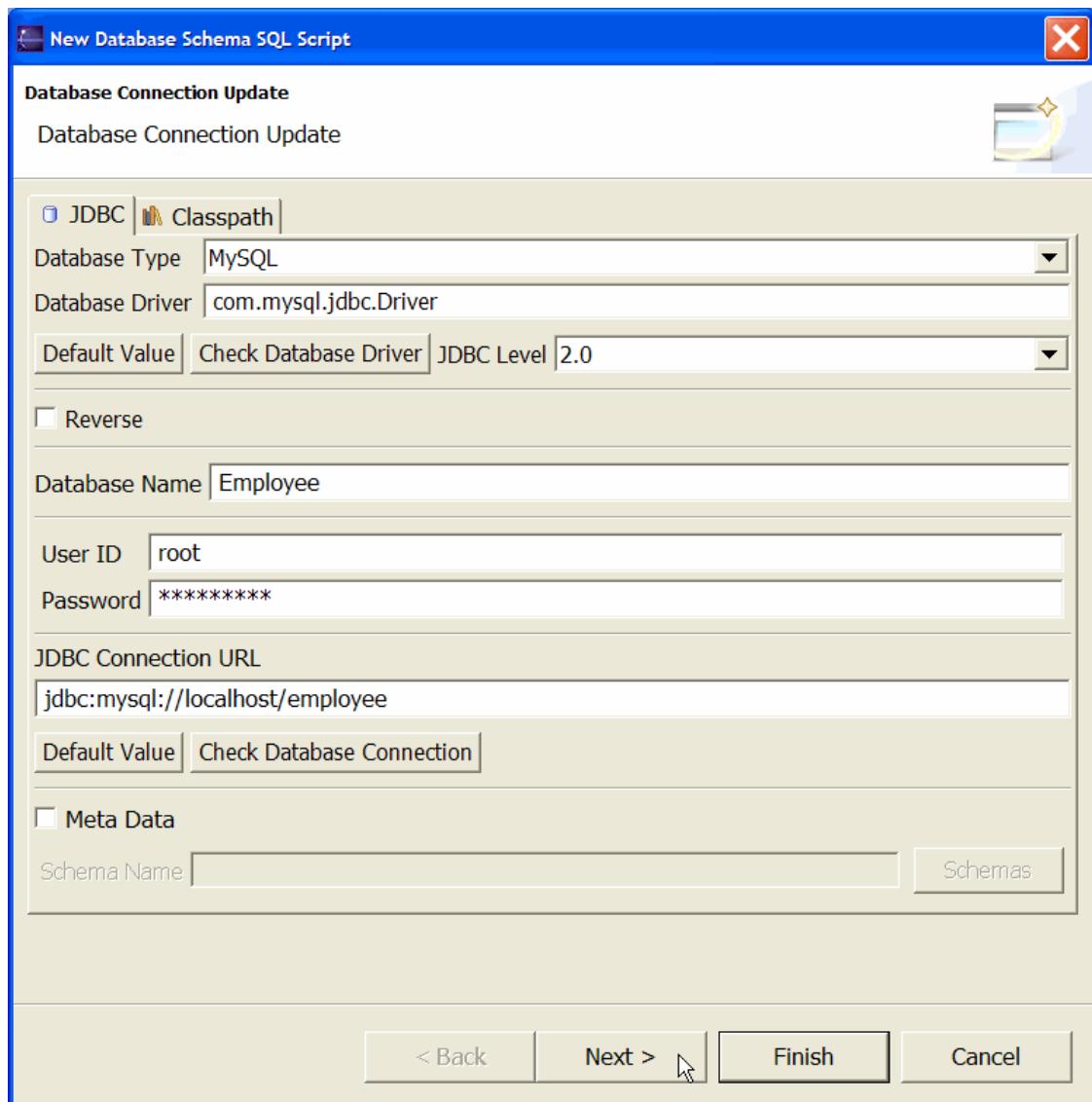
New->Other->Database->Database Schema SQL Script

The selection is contextual.

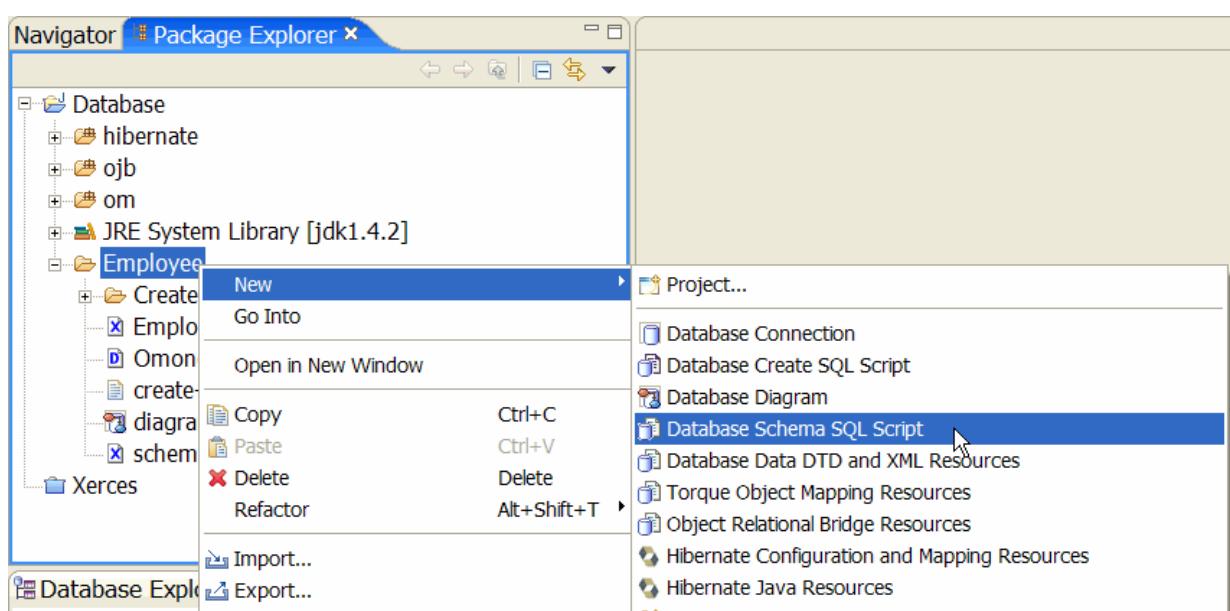
It means that if you select an existing Database Object:



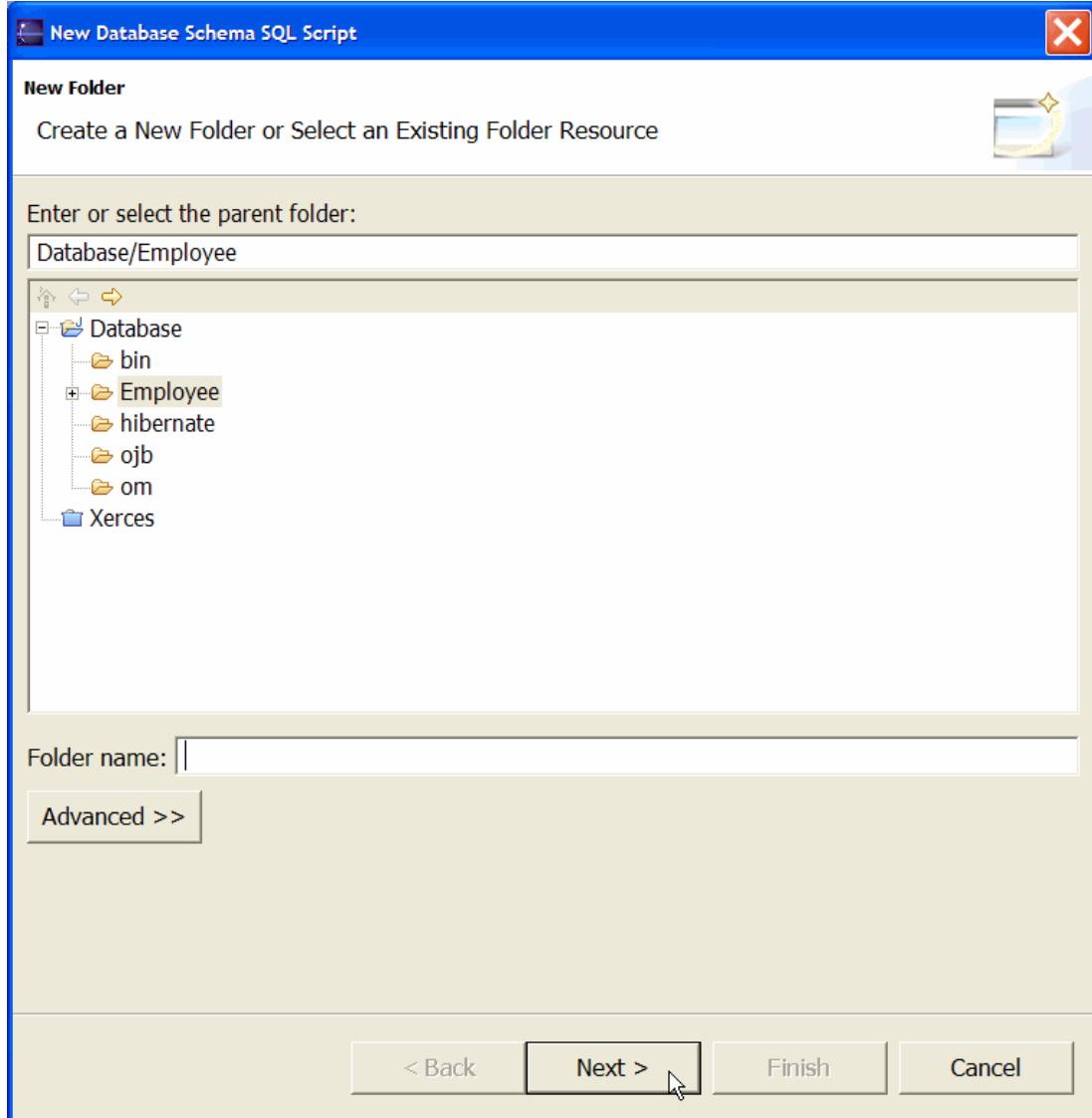
The wizard will open the parent Database Connection in update mode.



Otherwise :



A new Database Connection wizard will be opened starting with a [New Folder](#) selector.



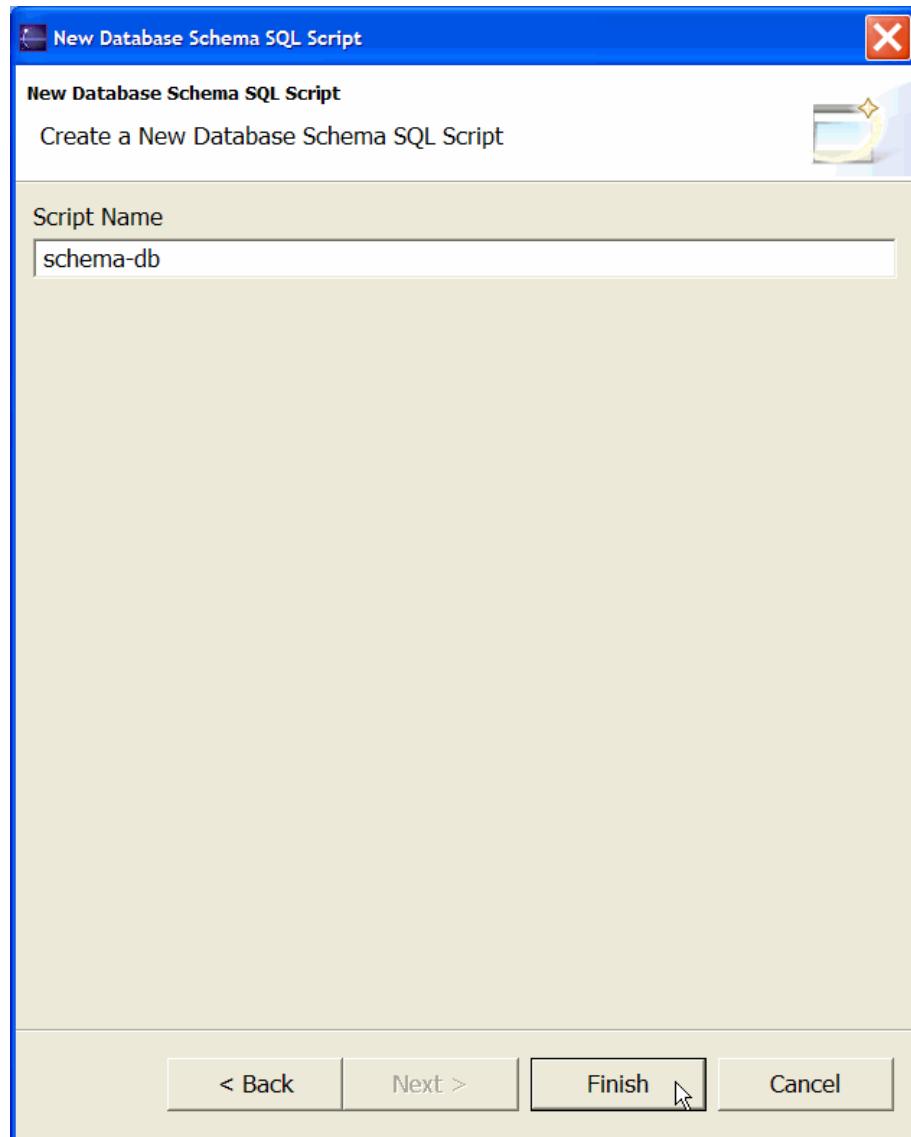
In Database Connection Update mode this wizard has four pages :

- [Database Connection Update](#)
- [Database Schema Update](#)
- [Database Schema DTD Update](#)
- [New Database Schema SQL Script](#)

In Database Connection Create mode this wizard has five pages :

- [New Folder](#)
- [New Database Connection](#)
- [New Database Schema](#)
- [New Database Schema DTD](#)
- [New Database Schema SQL Script](#)

2.1. Database Schema SQL Script Page



2.1.1. Script Name

Script is the name of your new SQL script.

Each resource should have a unique name in the targeted selected workspace folder or project root.

This data is not related to any Meta Data information.

This name will be the file name of the generated SQL file.

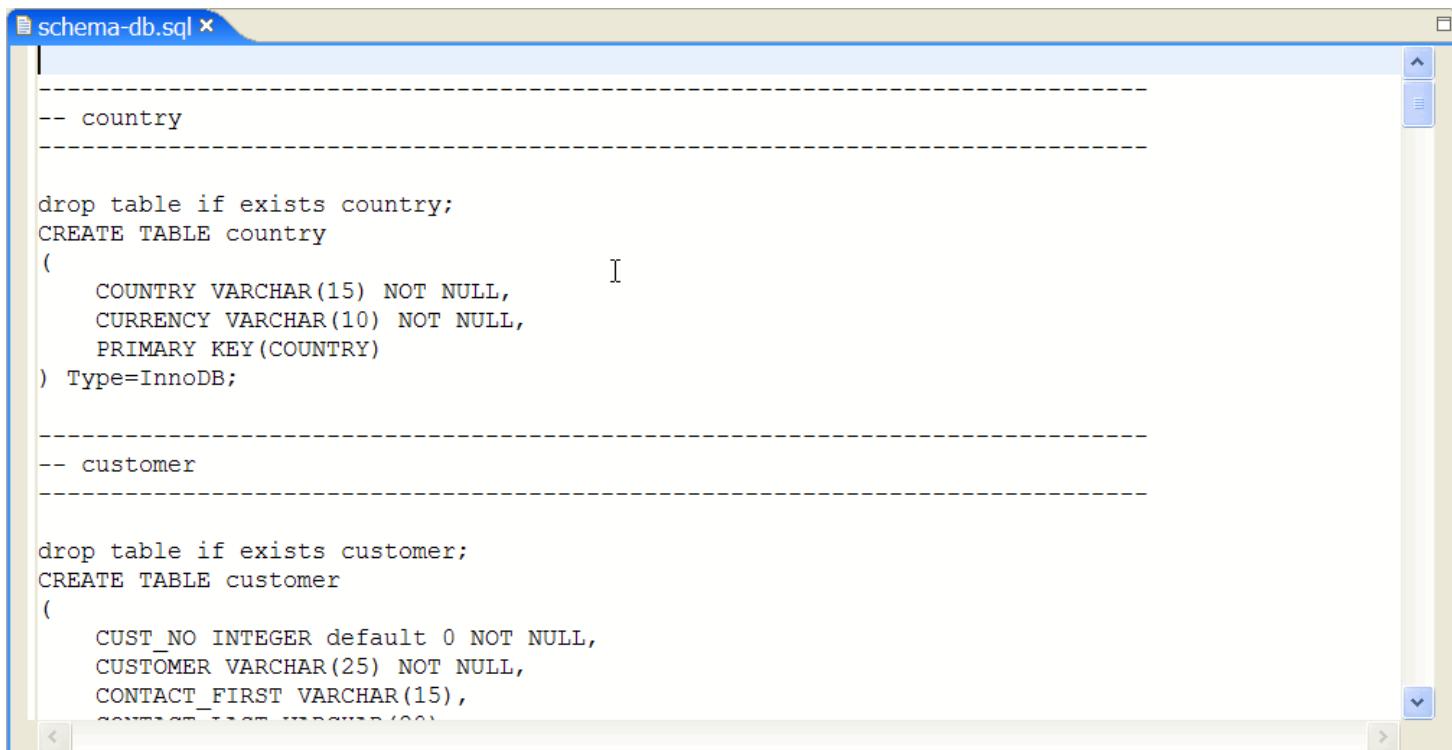
A Database Schema script is an SQL file, its file extension is :

- .sql

This file is referenced in your current [Database Connection](#).

2.2. Finish

The script generation of the new Database Schema SQL Script will be processed and created.

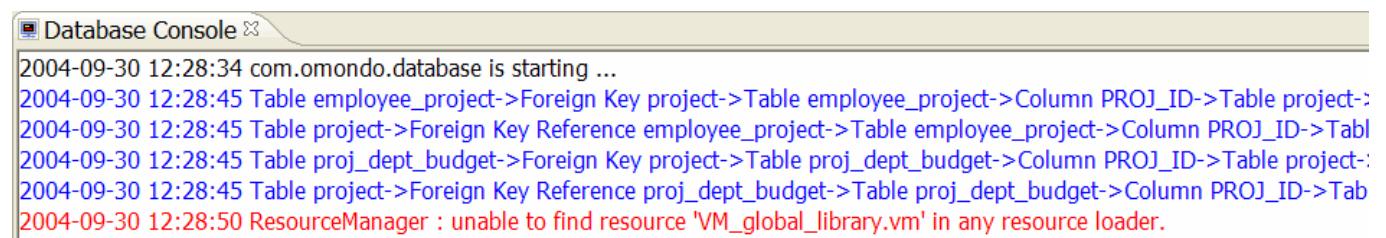


```
-- country
-----
drop table if exists country;
CREATE TABLE country
(
    COUNTRY VARCHAR(15) NOT NULL,
    CURRENCY VARCHAR(10) NOT NULL,
    PRIMARY KEY(COUNTRY)
) Type=InnoDB;

-- customer
-----

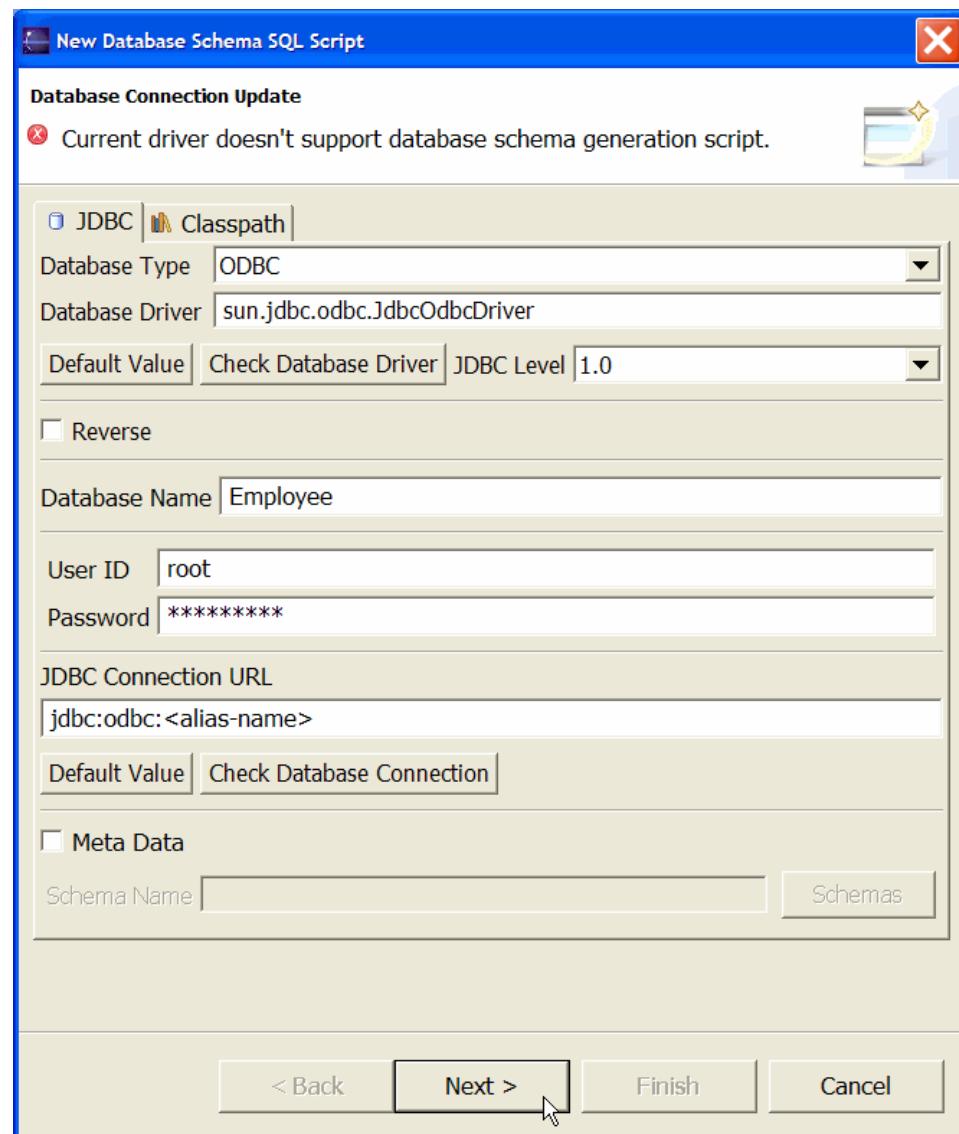
drop table if exists customer;
CREATE TABLE customer
(
    CUST_NO INTEGER default 0 NOT NULL,
    CUSTOMER VARCHAR(25) NOT NULL,
    CONTACT_FIRST VARCHAR(15),
    CONTACT_LAST VARCHAR(25)
```

Detailed information is available in the [DatabaseConsole](#) window.



```
2004-09-30 12:28:34 com.omondo.database is starting ...
2004-09-30 12:28:45 Table employee_project->Foreign Key project->Table employee_project->Column PROJ_ID->Table project-:
2004-09-30 12:28:45 Table project->Foreign Key Reference employee_project->Table employee_project->Column PROJ_ID->Tabl
2004-09-30 12:28:45 Table proj_dept_budget->Foreign Key project->Table proj_dept_budget->Column PROJ_ID->Table project-:
2004-09-30 12:28:45 Table project->Foreign Key Reference proj_dept_budget->Table proj_dept_budget->Column PROJ_ID->Tab
2004-09-30 12:28:50 ResourceManager : unable to find resource 'VM_global_library.vm' in any resource loader.
```

Some databases don't support schema generation.
In this case a message is displayed in the Database Connection page.

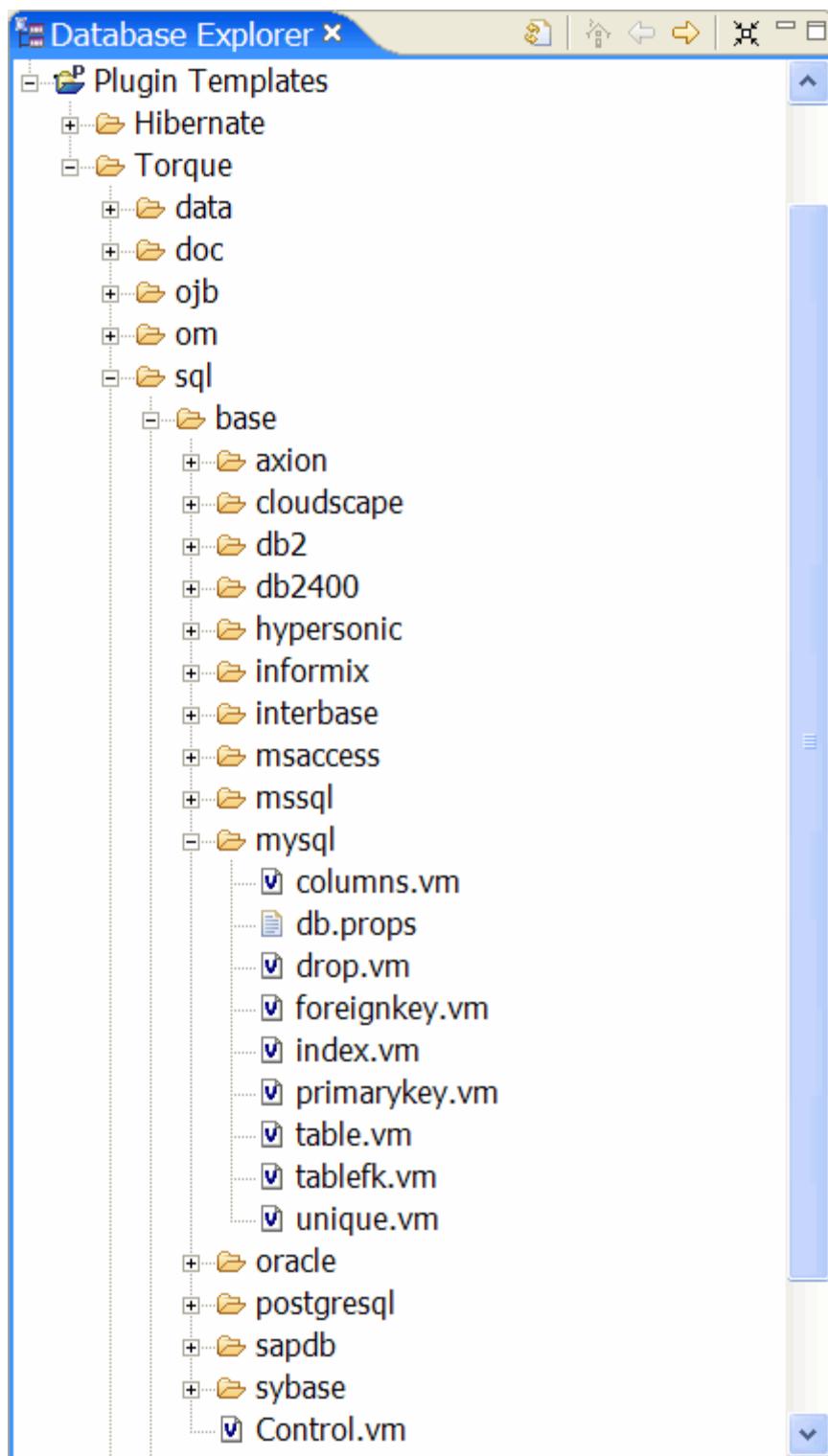


3. *Templates*

This wizard is [Template](#) based.

Below your [Plugin Templates](#) or [MyTemplates](#) you should see the following directory:

Torque/sql/base



These templates are [Velocity](#) based and use the [Torque Model API](#).

Database DTD and XML Ressources

1. [Introduction](#)
2. [Database Data DTD and XML Resources](#)
 1. [Database Data DTD and XML Resources Page](#)
 1. [Database Data DTD Name](#)
 2. [Database Data XML Name](#)
 2. [Finish](#)
3. [Templates](#)
4. [Additional Resources](#)

1. Introduction

The purpose of this chapter is to show how to generate a Data DTD and XML Resources.

This tool will produce two files :

- a DTD who describes your Data
- an XML file who contains your Data.

The Datas are extracted from your Database.

2. Database Data DTD and XML Resources

To start the Database Data DTD and XML Resources wizard, select :

File->New->Other->Database->Database Data DTD and XML Resources

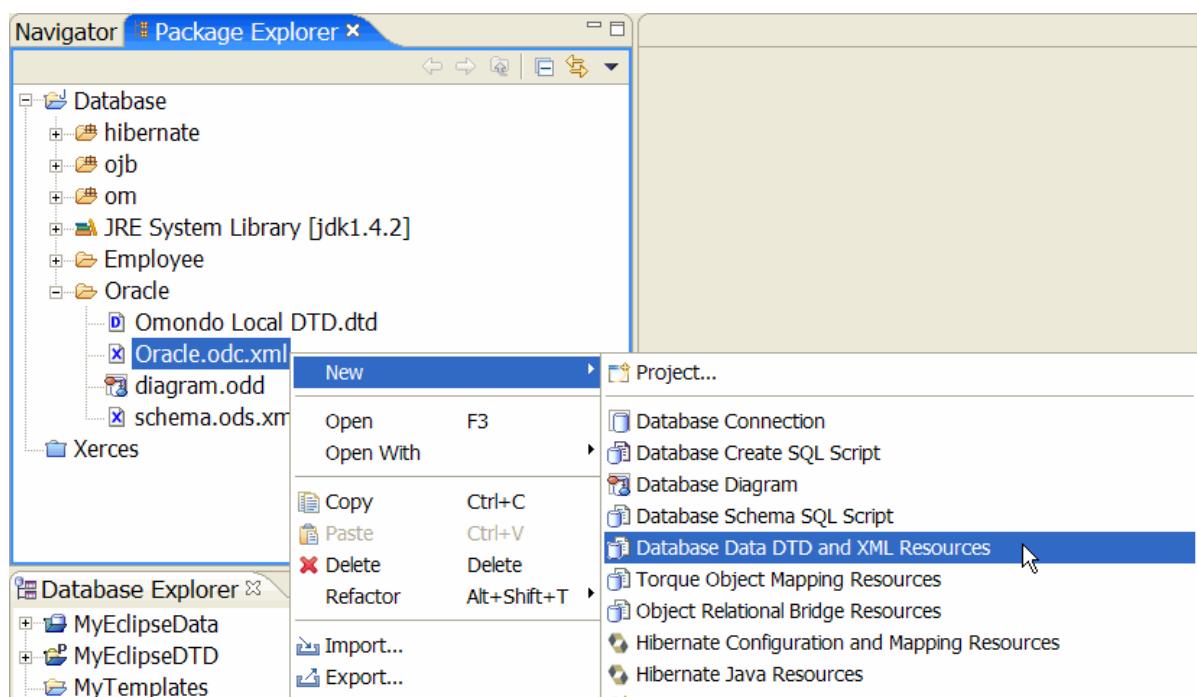
or

Select an object in the Package Explorer View or the Navigator View then right-click and select :

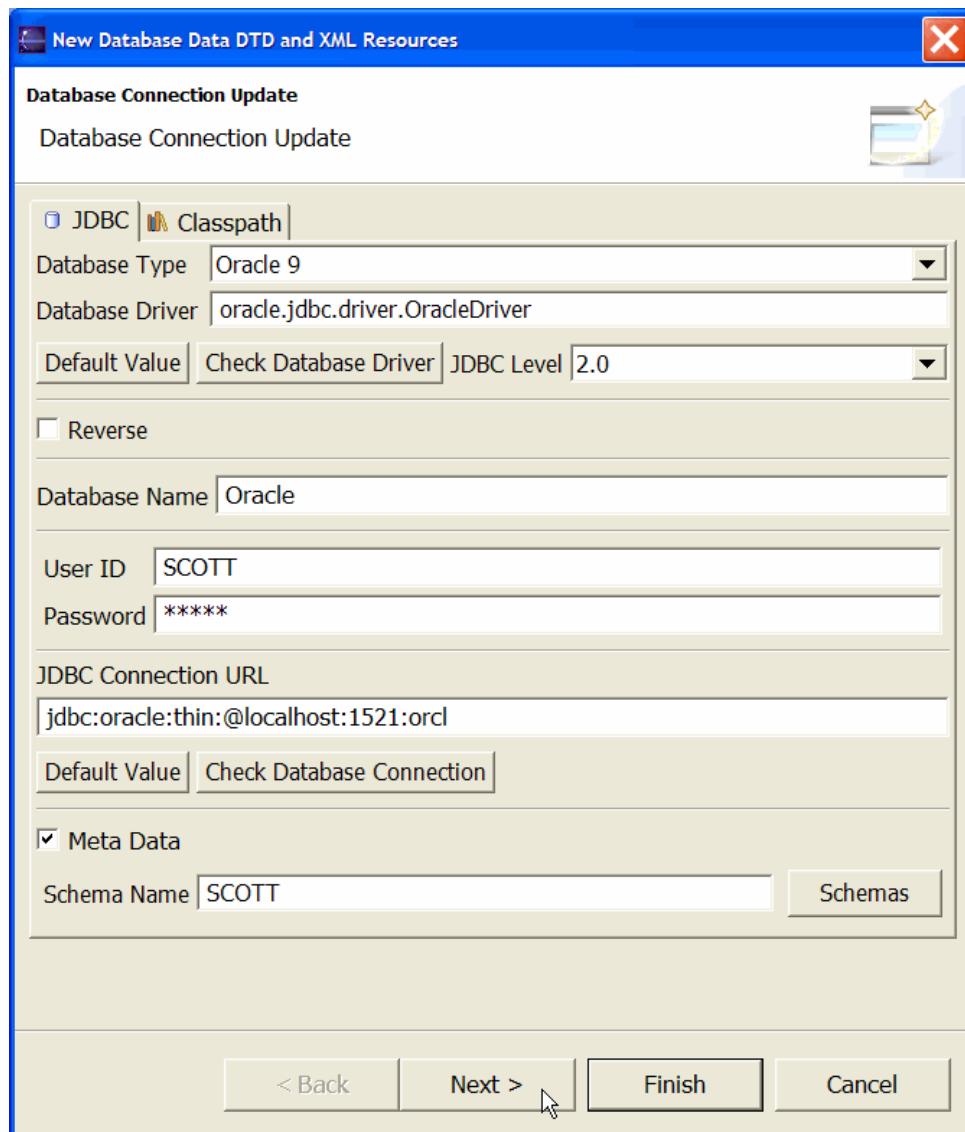
New->Other->Database->Database Data DTD and XML Resources

The selection is contextual.

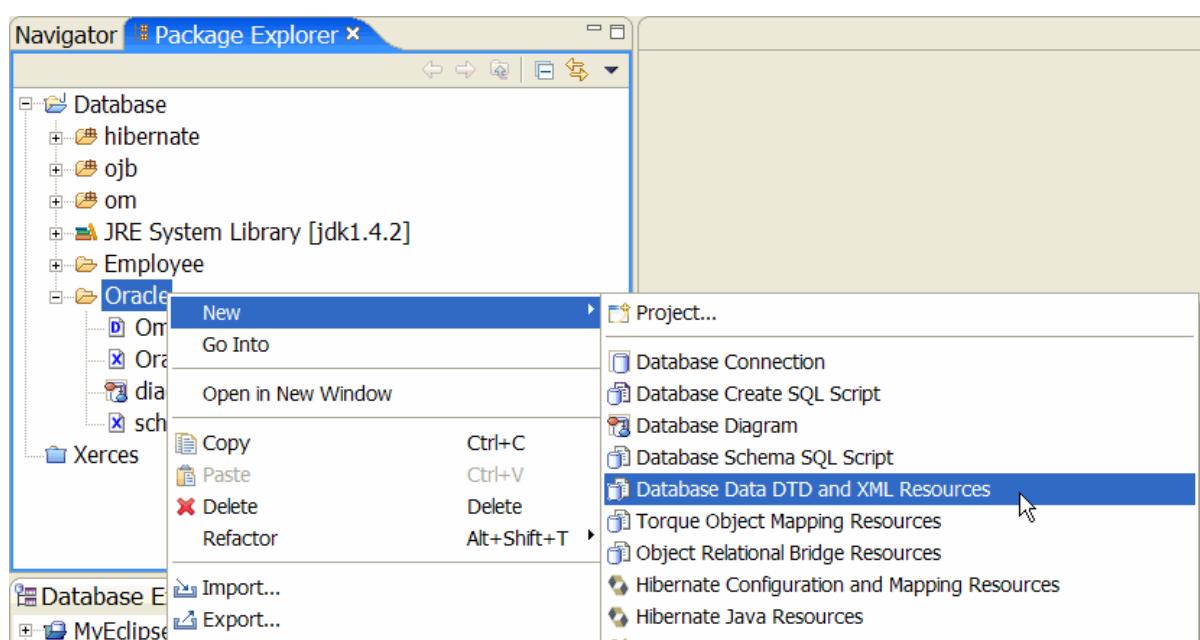
It means that if you select an existing Database Object:



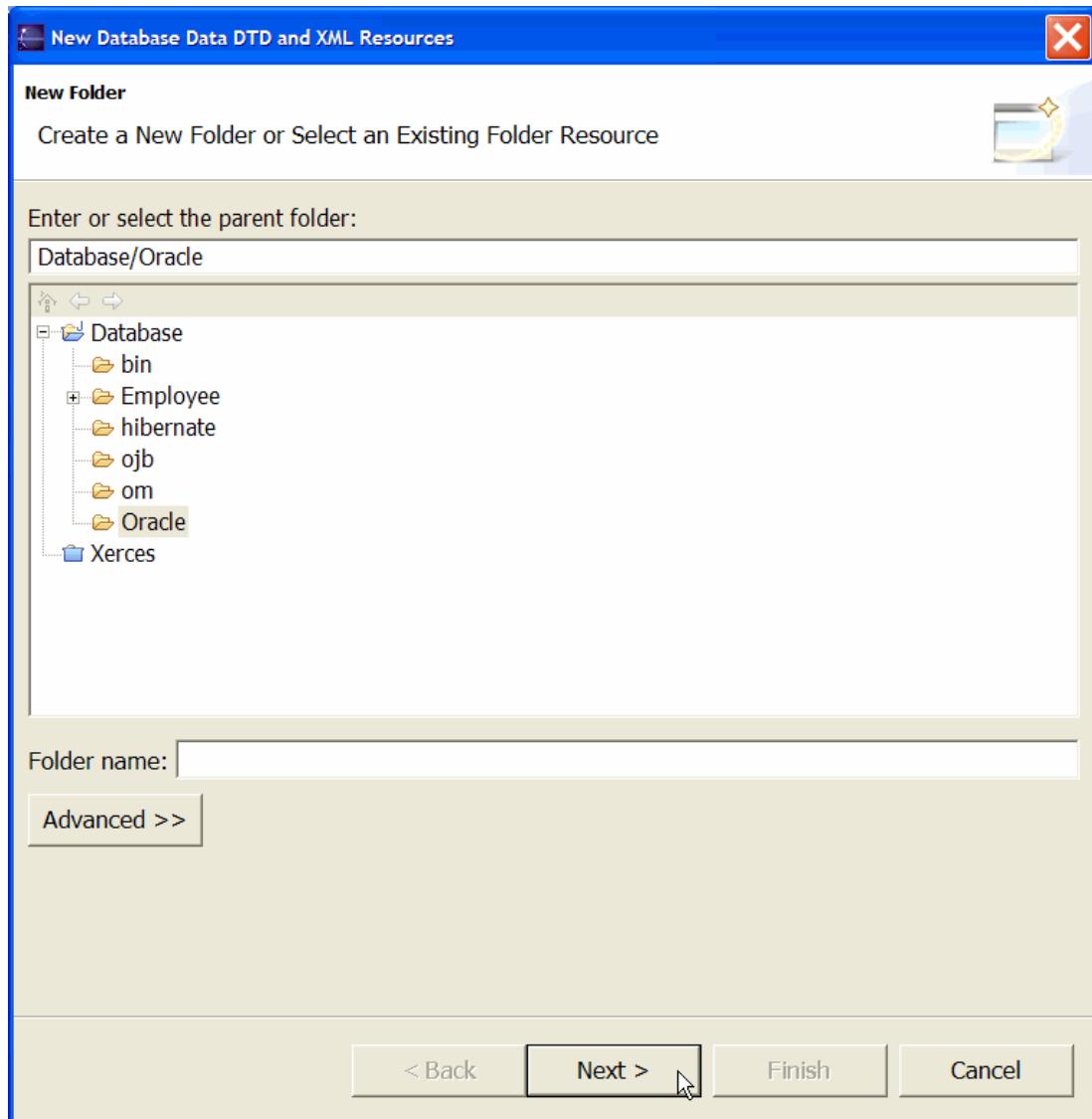
The wizard will open the parent Database Connection in update mode.



Otherwise :



A new Database Connection wizard will be opened starting with a [New Folder](#) selector.



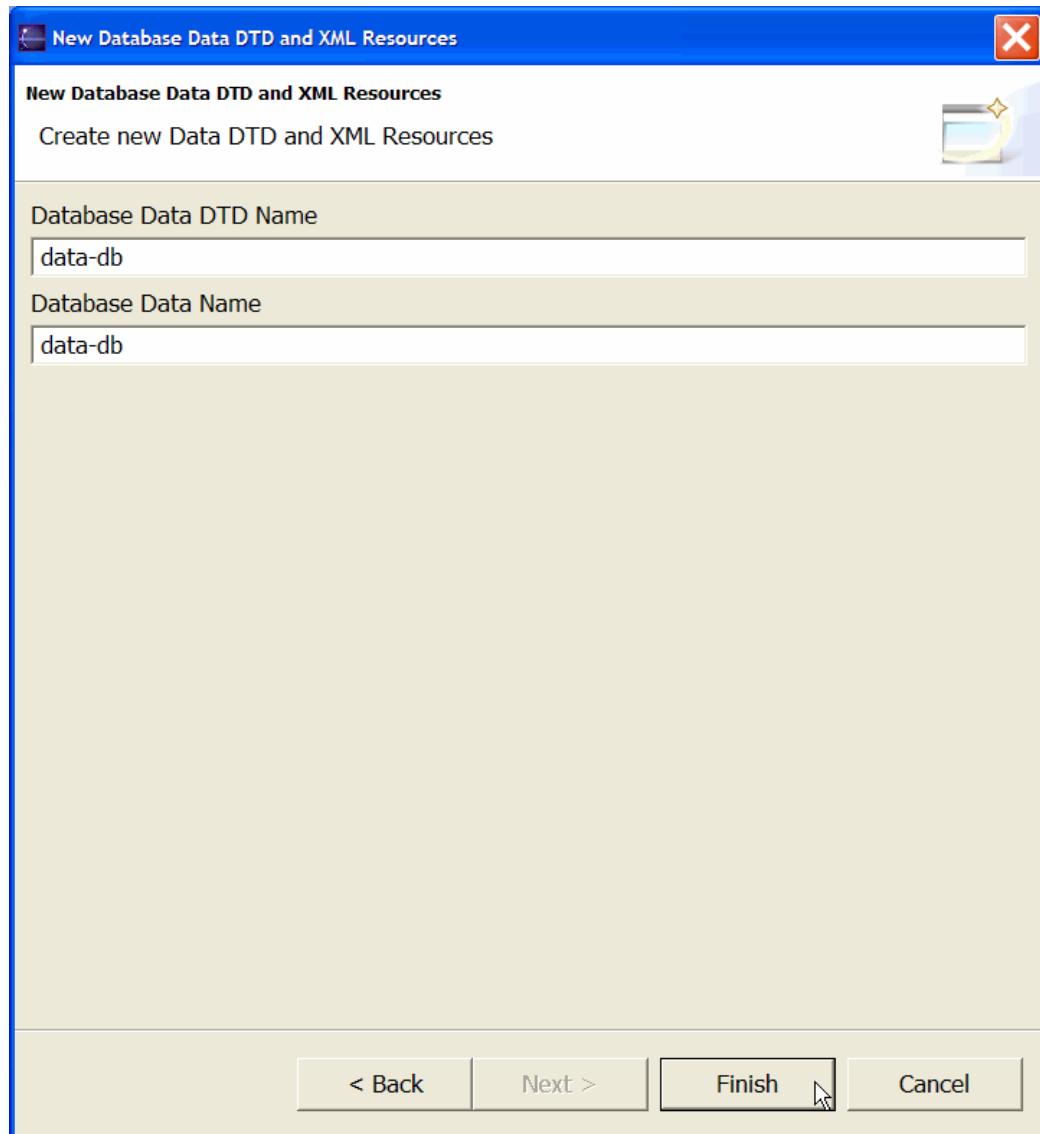
In Database Connection Update mode this wizard has four pages :

- [Database Connection Update](#)
- [Database Schema Update](#)
- [Database Schema DTD Update](#)
- [New Database Data DTD and XML Resources](#)

In Database Connection Create mode this wizard has five pages :

- [New Folder](#)
- [New Database Connection](#)
- [New Database Schema](#)
- [New Database Schema DTD](#)
- [New Database Data DTD and XML Resources](#)

2.1. Database Data DTD and XML Resources Page



2.1.1. Database Data DTD Name

Database Data DTD Name is the name of your new Data DTD.

Each resource should have a unique name in the targeted selected workspace folder or project root.

This data is not related to any Meta Data informations.

This name will be the file name of the generated Data DTD file.

A Database Data DTD is a DTD file, its file extension is :

- .dtd

This file is referenced in your current Database Data file.

2.1.1.Database Data XML Name

Database Data XML Name is the name of your new Database XML file.

Each resource should have a unique name in the targeted selected workspace folder or project root.

This data is not related to any Meta Data informations.

This name will be the file name of the generated Database Data XML file.

A Database Data XML is an XML file, its file extension could be :

- .odx.xml
- .odx

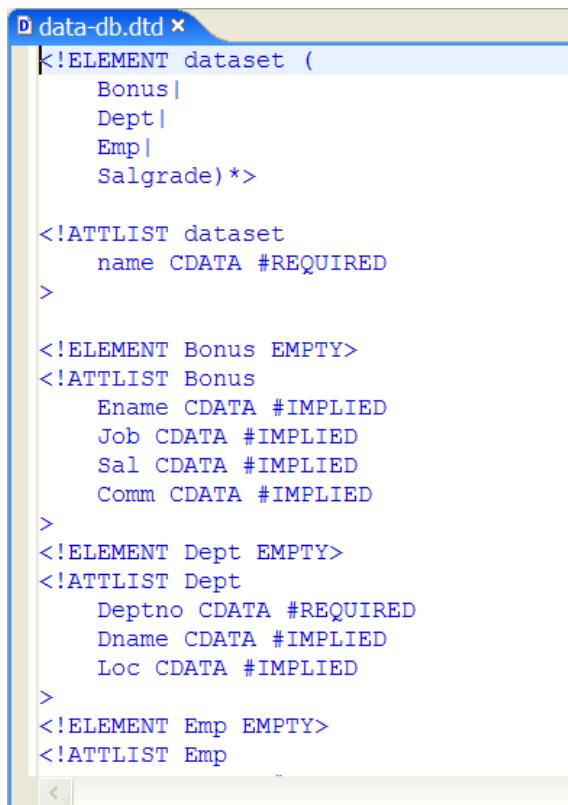
This file extension is managed by the [Save Policy](#) Preference.

This file is referenced in your current [Database Connection](#).

2.2. Finish

Two files will be generated.

A DTD file will be generated, this file describes your XML Data.



```

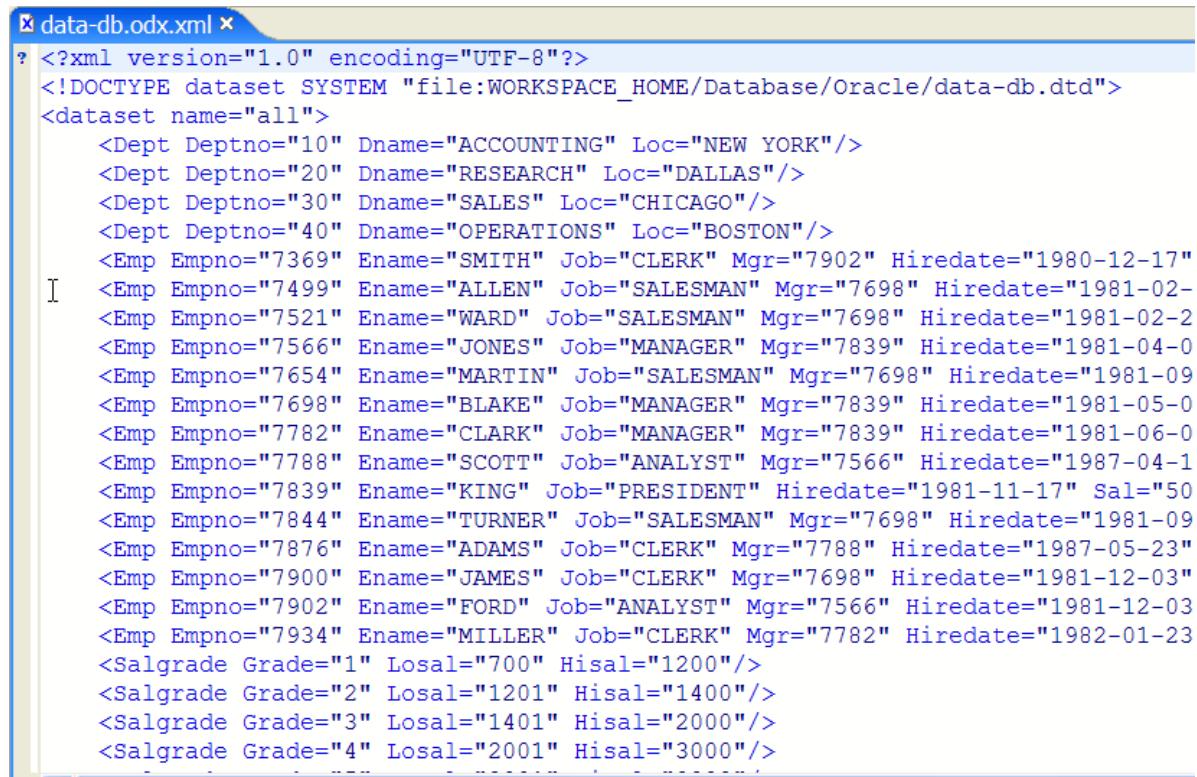
data-db.dtd x
<!ELEMENT dataset (
    Bonus|
    Dept|
    Emp|
    Salgrade)*>

<!ATTLIST dataset
      name CDATA #REQUIRED
>

<!ELEMENT Bonus EMPTY>
<!ATTLIST Bonus
      Ename CDATA #IMPLIED
      Job CDATA #IMPLIED
      Sal CDATA #IMPLIED
      Comm CDATA #IMPLIED
>
<!ELEMENT Dept EMPTY>
<!ATTLIST Dept
      Deptno CDATA #REQUIRED
      Dname CDATA #IMPLIED
      Loc CDATA #IMPLIED
>
<!ELEMENT Emp EMPTY>
<!ATTLIST Emp
      ...
>

```

A XML file will be generated, this file contains your extracted data.

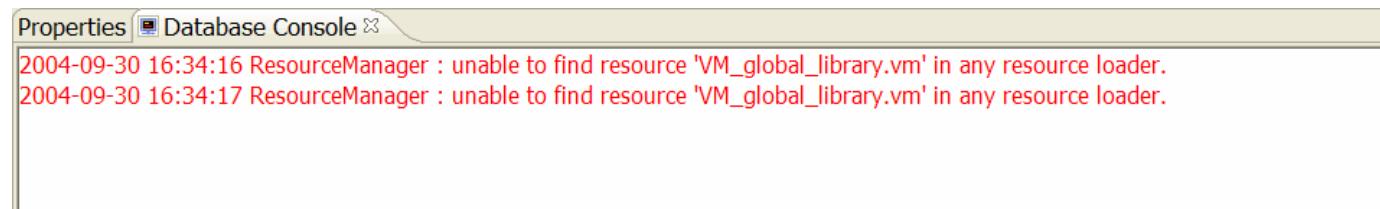


```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dataset SYSTEM "file:WORKSPACE_HOME/Database/Oracle/data-db.dtd">
<dataset name="all">
  <Dept Deptno="10" Dname="ACCOUNTING" Loc="NEW YORK"/>
  <Dept Deptno="20" Dname="RESEARCH" Loc="DALLAS"/>
  <Dept Deptno="30" Dname="SALES" Loc="CHICAGO"/>
  <Dept Deptno="40" Dname="OPERATIONS" Loc="BOSTON"/>
  <Emp Empno="7369" Ename="SMITH" Job="CLERK" Mgr="7902" Hiredate="1980-12-17"
    I
  <Emp Empno="7499" Ename="ALLEN" Job="SALESMAN" Mgr="7698" Hiredate="1981-02-
  <Emp Empno="7521" Ename="WARD" Job="SALESMAN" Mgr="7698" Hiredate="1981-02-2
  <Emp Empno="7566" Ename="JONES" Job="MANAGER" Mgr="7839" Hiredate="1981-04-0
  <Emp Empno="7654" Ename="MARTIN" Job="SALESMAN" Mgr="7698" Hiredate="1981-09
  <Emp Empno="7698" Ename="BLAKE" Job="MANAGER" Mgr="7839" Hiredate="1981-05-0
  <Emp Empno="7782" Ename="CLARK" Job="MANAGER" Mgr="7839" Hiredate="1981-06-0
  <Emp Empno="7788" Ename="SCOTT" Job="ANALYST" Mgr="7566" Hiredate="1987-04-1
  <Emp Empno="7839" Ename="KING" Job="PRESIDENT" Hiredate="1981-11-17" Sal="50
  <Emp Empno="7844" Ename="TURNER" Job="SALESMAN" Mgr="7698" Hiredate="1981-09
  <Emp Empno="7876" Ename="ADAMS" Job="CLERK" Mgr="7788" Hiredate="1987-05-23"
  <Emp Empno="7900" Ename="JAMES" Job="CLERK" Mgr="7698" Hiredate="1981-12-03"
  <Emp Empno="7902" Ename="FORD" Job="ANALYST" Mgr="7566" Hiredate="1981-12-03
  <Emp Empno="7934" Ename="MILLER" Job="CLERK" Mgr="7782" Hiredate="1982-01-23
  <Salgrade Grade="1" Losal="700" Hisal="1200"/>
  <Salgrade Grade="2" Losal="1201" Hisal="1400"/>
  <Salgrade Grade="3" Losal="1401" Hisal="2000"/>
  <Salgrade Grade="4" Losal="2001" Hisal="3000"/>
  . . .

```

Detailed information is available in the [DatabaseConsole](#) window.

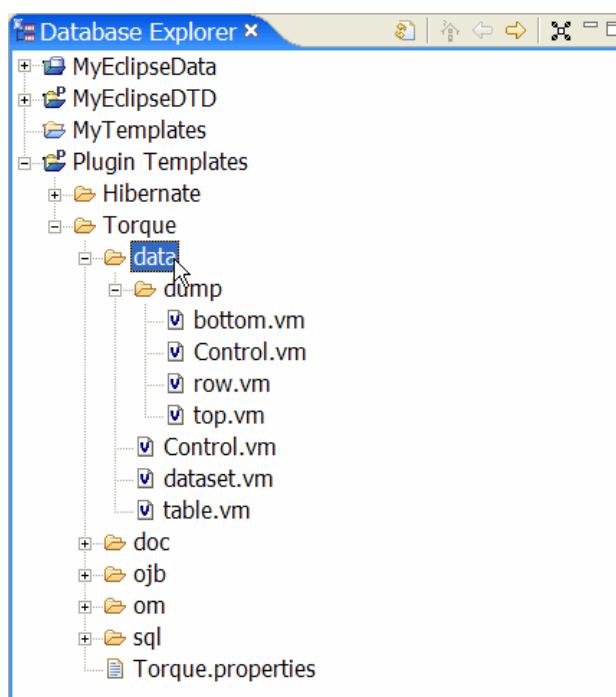


3. Templates

This wizard is [Template](#) based.

Below your [Plugin Templates](#) or [MyTemplates](#) you should see the following directory:

Torque/sql/data



Torque/sql/data contains the Database Data DTD Templates.

Torque/sql/data/dump contains the Database Data XML Templates.

These templates are [Velocity](#) based and use the [Torque Model API](#).

4. Additional Resources

Here are several links provided for further Apache Velocity related information.

- [Apache Velocity Project](#)
- [Velocity Mailing Lists](#)

Forward

1. [Introduction](#)
2. [Forward](#)
 1. [SQL Execution Settings](#)
 2. [Finish](#)

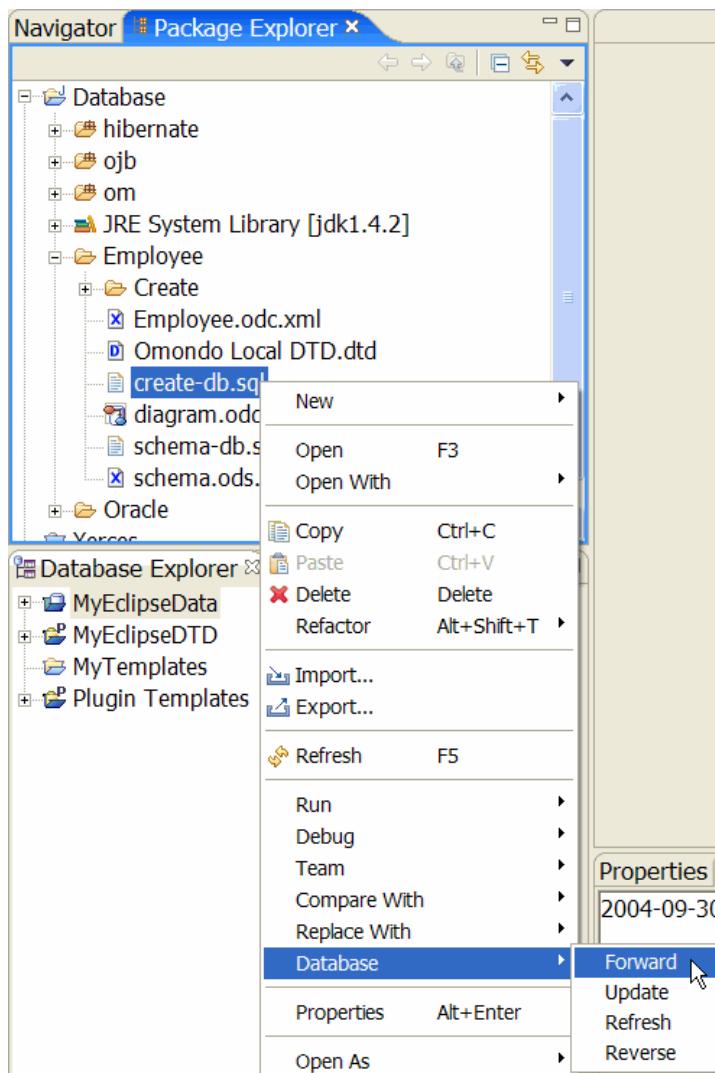
1. Introduction

The **Forward** command lets you run a SQL Script according to its Database Connection.

2. Forward

Select a Database SQL object in the Package Explorer View or the Navigator View then right-click and select :

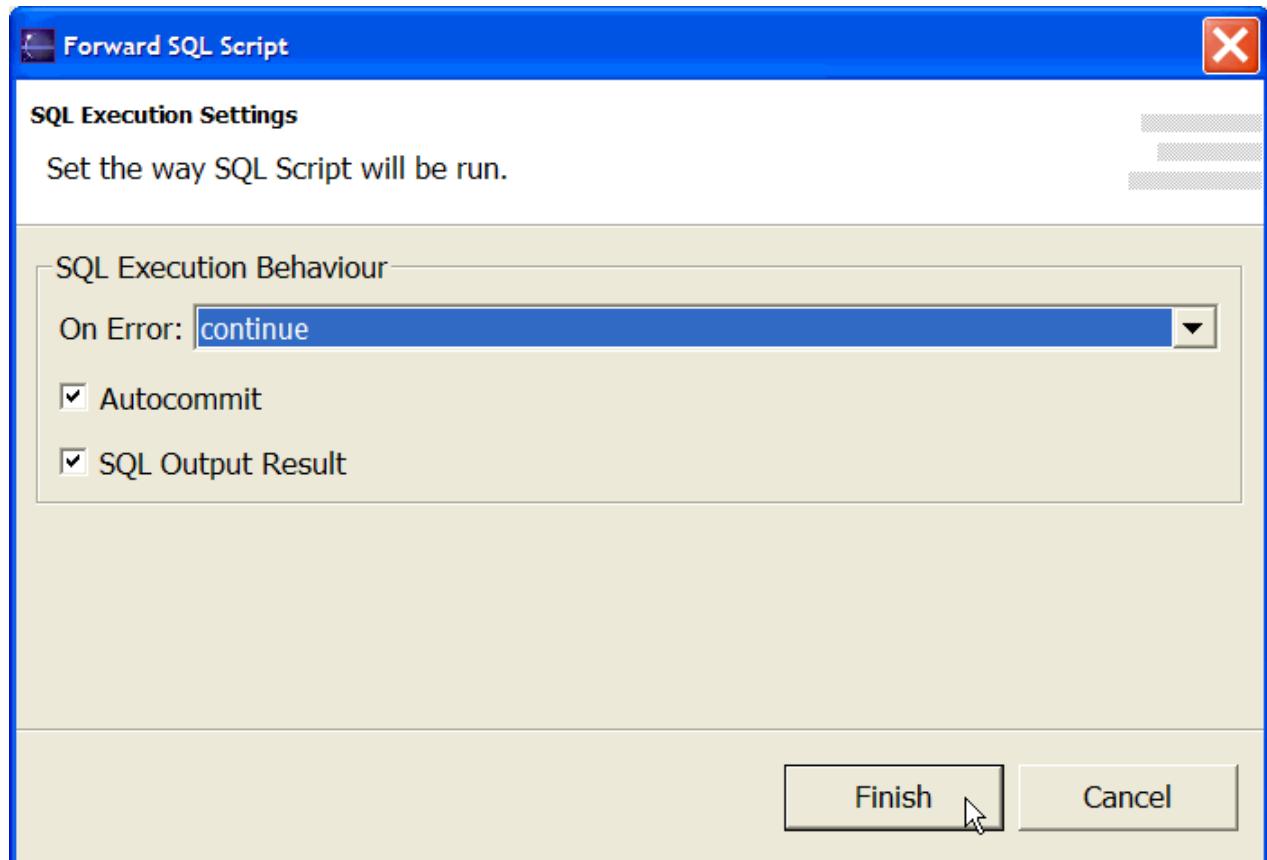
Database->Forward



The Forward wizard is composed of the following pages :

- [SQL Execution Settings](#)

2.1. SQL Execution Settings



The SQL Execution Settings page inherits the [Database SQL Global Preferences](#).

2.2. Finish

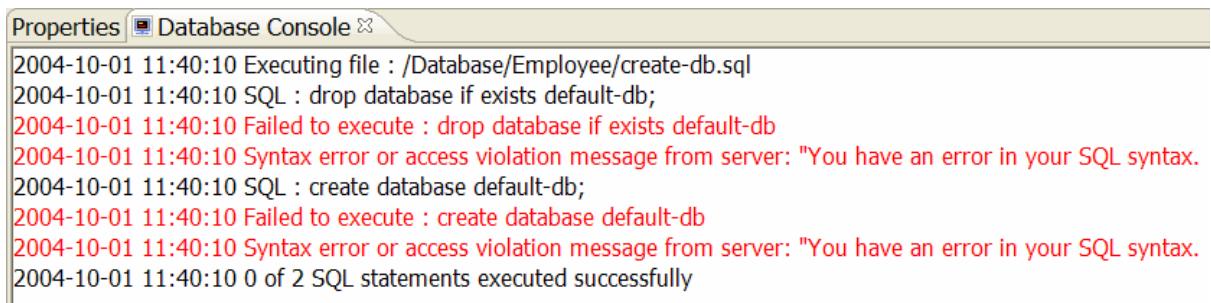
Once you Finish this page, your Database SQL script will be executed.

A report is displayed in the DatabaseConsole.

The screenshot shows the 'Database Console' window with the title 'Properties Database Console'. The log area displays the following output:

```
2004-10-01 11:38:12 Executing file : /Database/Employee/create-db.sql
2004-10-01 11:38:12 SQL : drop database if exists employee;
2004-10-01 11:38:13 0 rows affected
2004-10-01 11:38:13 SQL : create database employee;
2004-10-01 11:38:13 1 rows affected
2004-10-01 11:38:13 2 of 2 SQL statements executed successfully
```

If a problem arises, the DatabaseConsole can help.



The screenshot shows a window titled "Database Console" with the tab "Properties" selected. The main area displays a log of SQL statements and their execution results. The log entries are as follows:

```
2004-10-01 11:40:10 Executing file : /Database/Employee/create-db.sql
2004-10-01 11:40:10 SQL : drop database if exists default-db;
2004-10-01 11:40:10 Failed to execute : drop database if exists default-db
2004-10-01 11:40:10 Syntax error or access violation message from server: "You have an error in your SQL syntax.
2004-10-01 11:40:10 SQL : create database default-db;
2004-10-01 11:40:10 Failed to execute : create database default-db
2004-10-01 11:40:10 Syntax error or access violation message from server: "You have an error in your SQL syntax.
2004-10-01 11:40:10 0 of 2 SQL statements executed successfully
```

Index

1. [Introduction](#)
2. [Index](#)
 1. [Database Connection](#)
 2. [Folder](#)
 3. [Project](#)

1. Introduction

EclipseDatabase maintains an internal Index to manage the link between Database Files and their Database Connection.

As we saw in the [Database Connection UML Diagram](#), a Database Connection references various files :

- Database Schema
- Database Diagram
- Database SQL Script
- Database Data

It means that from a Database Connection file we can navigate to all its dependencies.

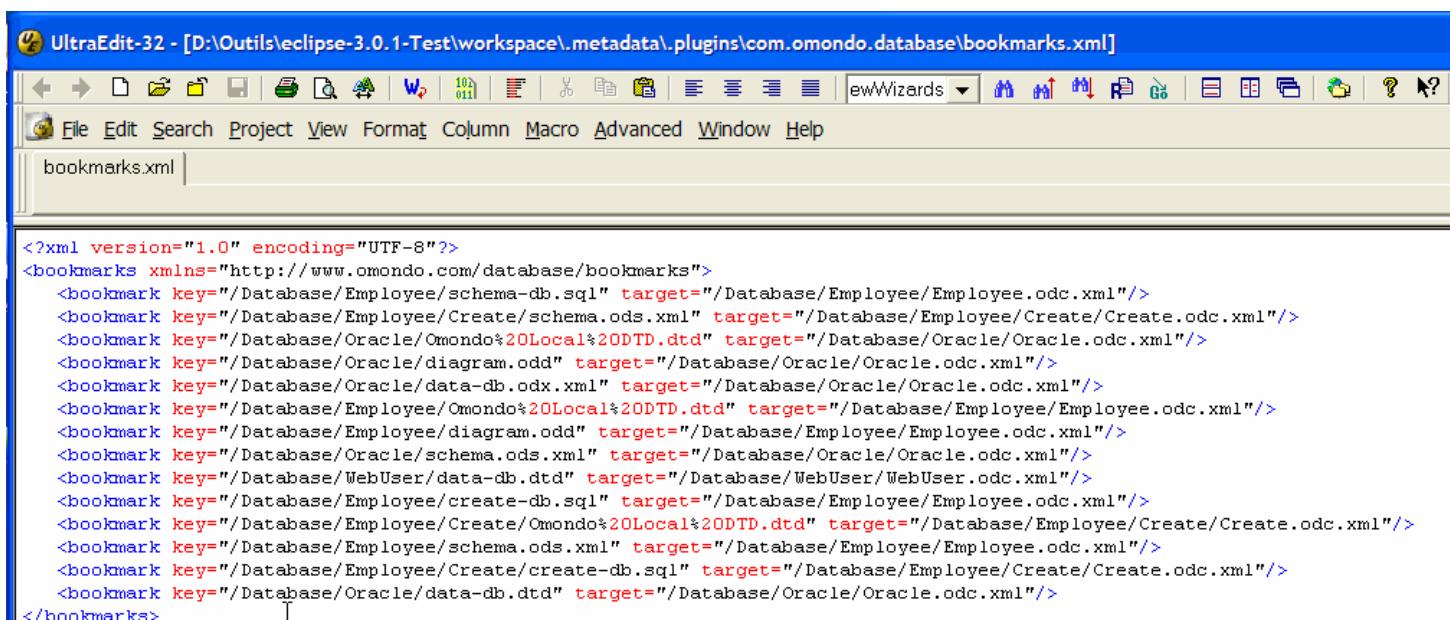
If we want to get the parent Database Connection from a Database Schema DTD, the task is more complex.

The Database Schema DTD doesn't know which is its parent Database Schema.

The Database Schema doesn't know which is its parent Database Connection.

Index fills this gap.

2. Index



The screenshot shows the UltraEdit-32 text editor interface with the title bar "UltraEdit-32 - [D:\Outils\eclipse-3.0.1-Test\workspace\.metadata\.plugins\com.omondo.database\bookmarks.xml]". The menu bar includes File, Edit, Search, Project, View, Format, Column, Macro, Advanced, Window, and Help. The toolbar has various icons for file operations like Open, Save, Find, and Copy. The status bar at the bottom shows the file path and some statistics. The main window displays the XML code for 'bookmarks.xml'.

```
<?xml version="1.0" encoding="UTF-8"?>
<bookmarks xmlns="http://www.omondo.com/database/bookmarks">
  <bookmark key="/Database/Employee/schema-db.sql" target="/Database/Employee/Employee.odc.xml"/>
  <bookmark key="/Database/Employee/Create/schema.ods.xml" target="/Database/Employee/Create/Create.xml"/>
  <bookmark key="/Database/Oracle/Omondo%20Local%20DTD.dtd" target="/Database/Oracle/Oracle.odc.xml"/>
  <bookmark key="/Database/Oracle/diagram.odd" target="/Database/Oracle/Oracle.odc.xml"/>
  <bookmark key="/Database/Oracle/data-db.odx.xml" target="/Database/Oracle/Oracle.odc.xml"/>
  <bookmark key="/Database/Employee/Omondo%20Local%20DTD.dtd" target="/Database/Employee/Employee.odc.xml"/>
  <bookmark key="/Database/Employee/diagram.odd" target="/Database/Employee/Employee.odc.xml"/>
  <bookmark key="/Database/Oracle/schema.ods.xml" target="/Database/Oracle/Oracle.odc.xml"/>
  <bookmark key="/Database/WebUser/data-db.dtd" target="/Database/WebUser/WebUser.odc.xml"/>
  <bookmark key="/Database/Employee/create-db.sql" target="/Database/Employee/Employee.odc.xml"/>
  <bookmark key="/Database/Employee/Create/Omondo%20Local%20DTD.dtd" target="/Database/Employee/Create/Create.xml"/>
  <bookmark key="/Database/Employee/schema.ods.xml" target="/Database/Employee/Employee.odc.xml"/>
  <bookmark key="/Database/Employee/Create/create-db.sql" target="/Database/Employee/Create/Create.xml"/>
  <bookmark key="/Database/Oracle/data-db.dtd" target="/Database/Oracle/Oracle.odc.xml"/>
</bookmarks>
```

Index is stored below the
%WORKSPACE_HOME%/.metadata/.plugins/com.omondo.database/bookmarks.xml file.

This Index is managed by the EclipseDatabase plugin.

However in some cases this index should be updated :

- Database files checkout
- Eclipse crash
- Out of sync resources in your workspace

Most of the time, the application will correctly manage the index :

- Database file creation
- Database file delete
- Database file **Refactor->Move** command

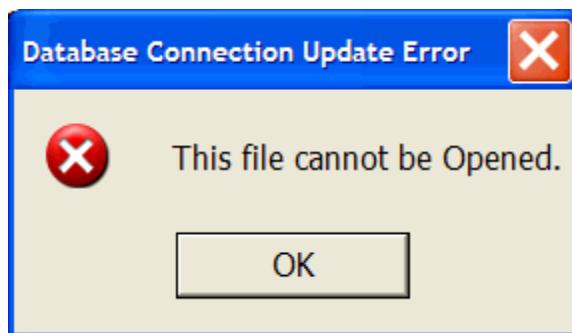
Even your Database files are outside the scope of your current project.

You can :

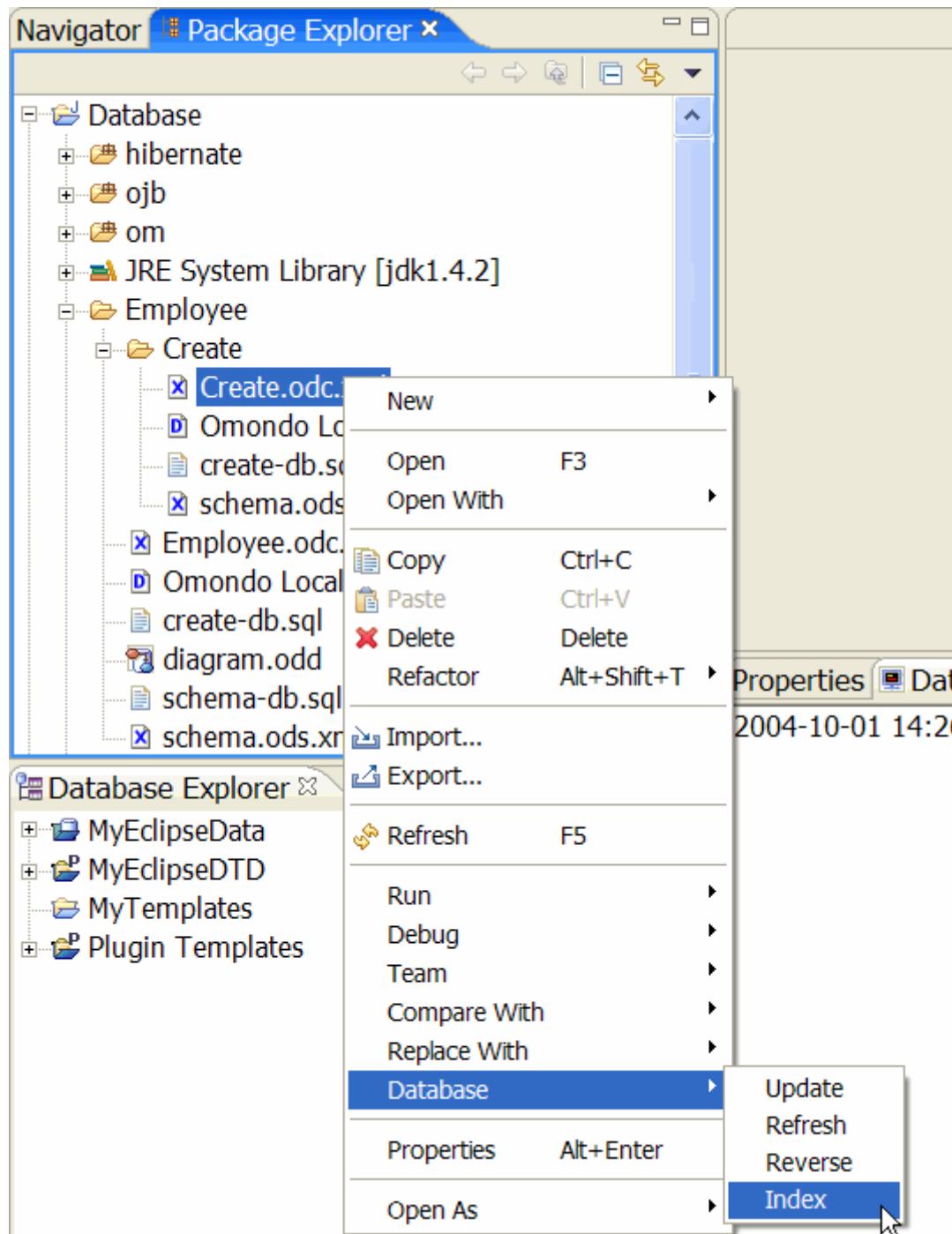
- Store a Database Schema DTD in one project
- Store its parent Database Schema in another project
- Store its Database Connection in another project

However, we do not recommend to do this.

If an Index is not found, EclipseDatabase will try to retrieve a Database Connection in the current project, otherwise will crash.



2.1. Database Connection



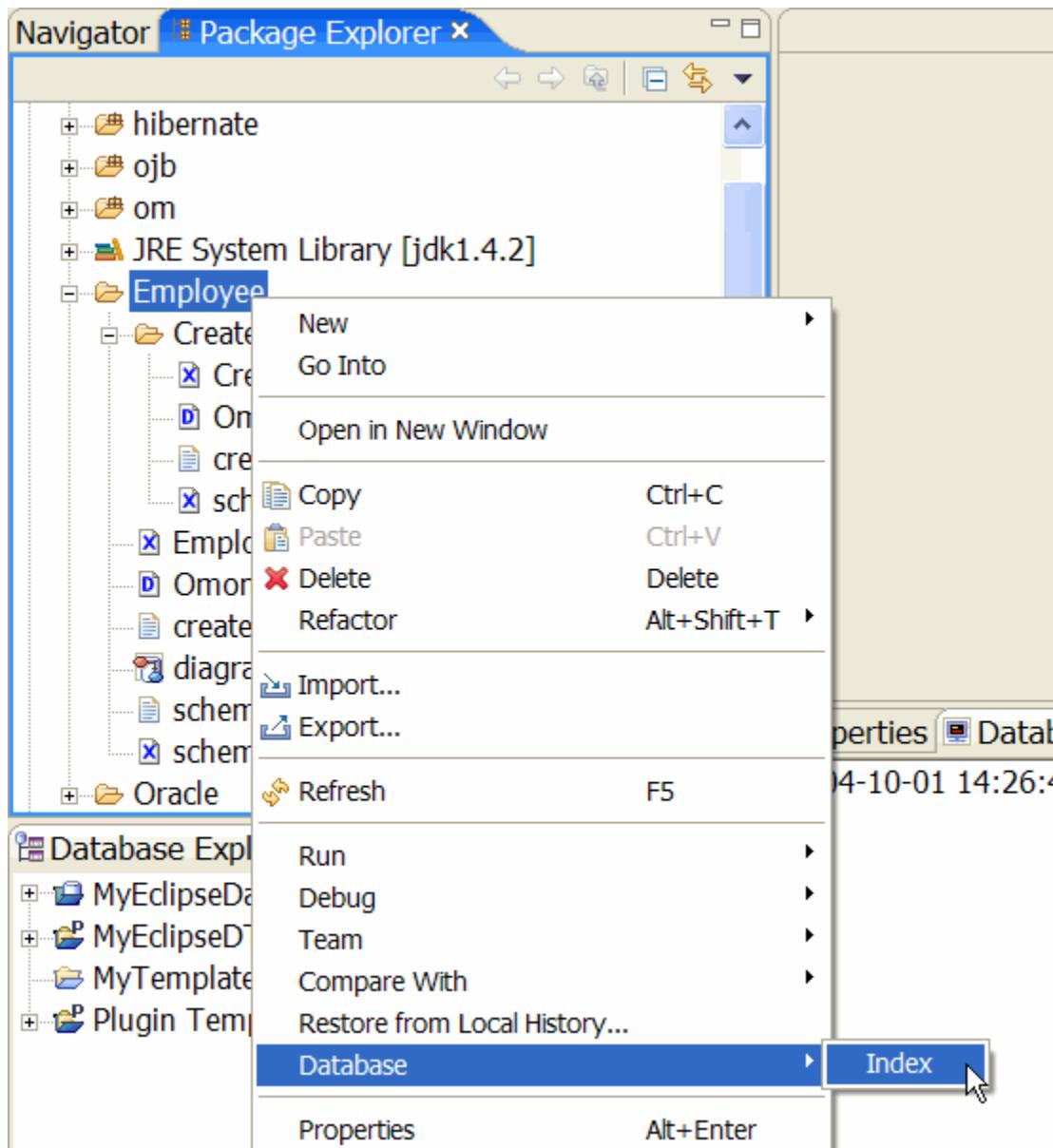
The Database Connection will be analysed and the Index will be rebuilt for this particular Database Connection.

The Database Schema will be parsed to analyse its Database Schema DTD.

The Database Data will be parsed to analyse their Database Data DTD.

This command is suitable when you checkout a Database Connection and its files from a team system.

2.2. Folder



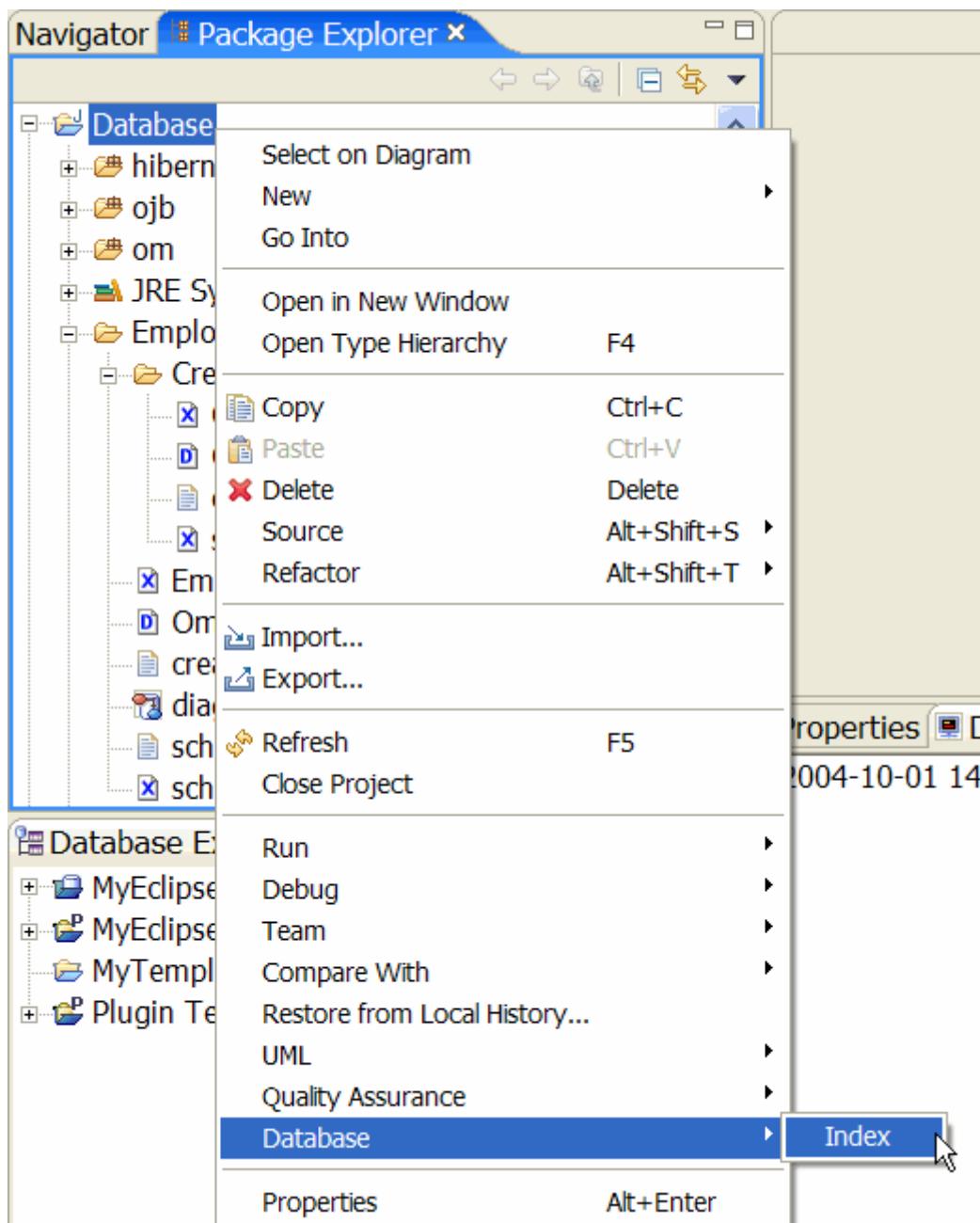
The folder and its sub-folders will be analysed to get all the contained Database Connections.

The array of Database Connections will be analysed and the Index will be rebuilt for this particular array of Database Connections.

The Database Schema will be parsed to analyse its Database Schema DTD.

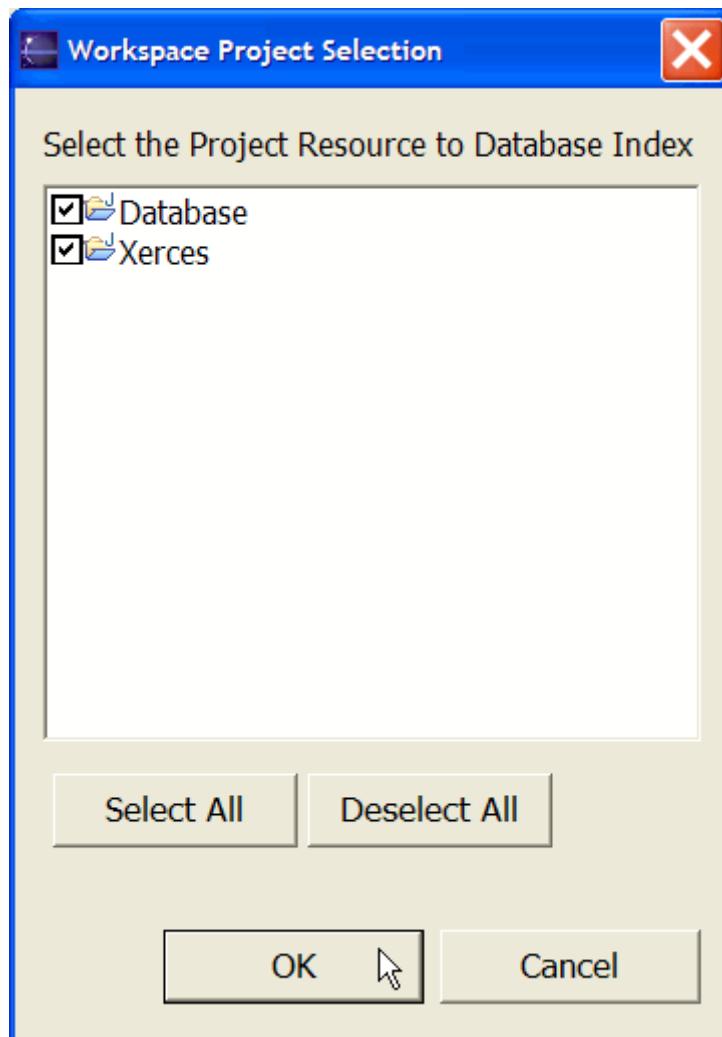
The Database Data will be parsed to analyse their Database Data DTDs.

2.3. Project



At the project level you have the opportunity to completely rebuild the Index.

In this case a project selector is displayed.



The index will be erased.

Closed projects will not be opened.

The projects and their sub-folders will be analysed to get all the contained Database Connections.

The array of Database Connections will be analysed and the Index will be rebuilt for this particular array of Database Connections.

The Database Schema will be parsed to analyse its Database Schema DTD.

The Database Data will be parsed to analyse their Database Data DTDs.

Modeling with the Database Diagram Explorer

1. [Introduction](#)
2. [Concept](#)
3. [Editor Commands](#)
 1. [Database Diagram Toolbar](#)
 2. [Database Schema Selector](#)
4. [Editor Preferences](#)
5. [Editor Properties](#)

1. *Introduction*

This chapter is designed to get you started on using the Database Diagram editor for Eclipse.

2. *Concept*

A data model is a conceptual representation of the data structures which include :

- The data objects
- The associations between data objects
- The rules which control operations on the objects

A data model is independent of hardware or software constraints.

The data model focuses on representing the data as the developer sees it in the "real world".

The data model is a bridge between :

- The concepts that make up real-world events and processes
- The physical representation of those concepts in a database

The Database Diagram editor will help you to follow the Database process :

- Analysis
- Conceptual Design
- Logical Design
- Physical Design
- Implementation

Database Diagram - Editor Command

1. Introduction
2. Database Diagram Toolbar
 1. Selection mode
 2. Zoom mode
 3. Create a table
 4. Create an association
 5. Create a note
 6. Create an indication
 7. Create a diagram link
 8. Create a text label

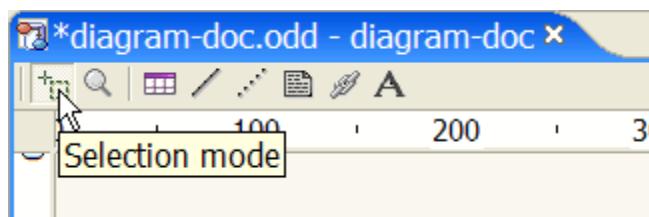
1. Introduction

This section describes the tools provided in order to build a Database Diagram.

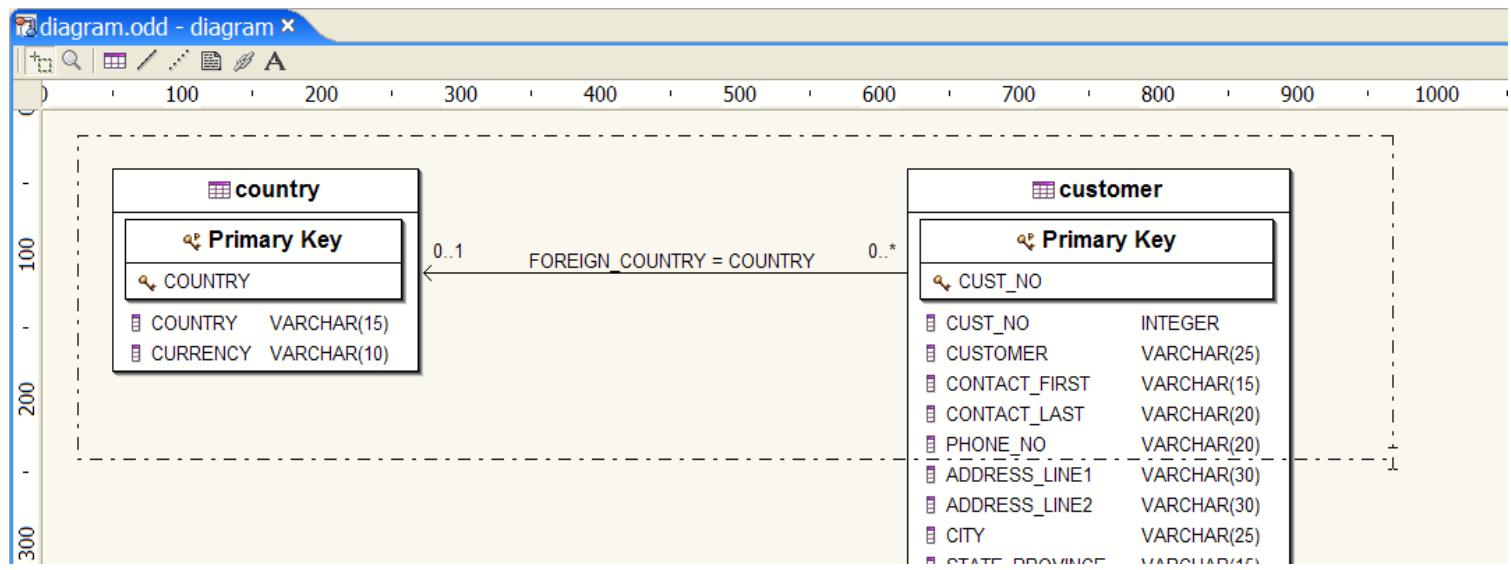
-  Selection mode
-  Zoom mode
-  Create a table
-  Create an association
-  Create an Indication
-  Create a note
-  Create a diagram link
-  Create a text label

2. Database Diagram Toolbar

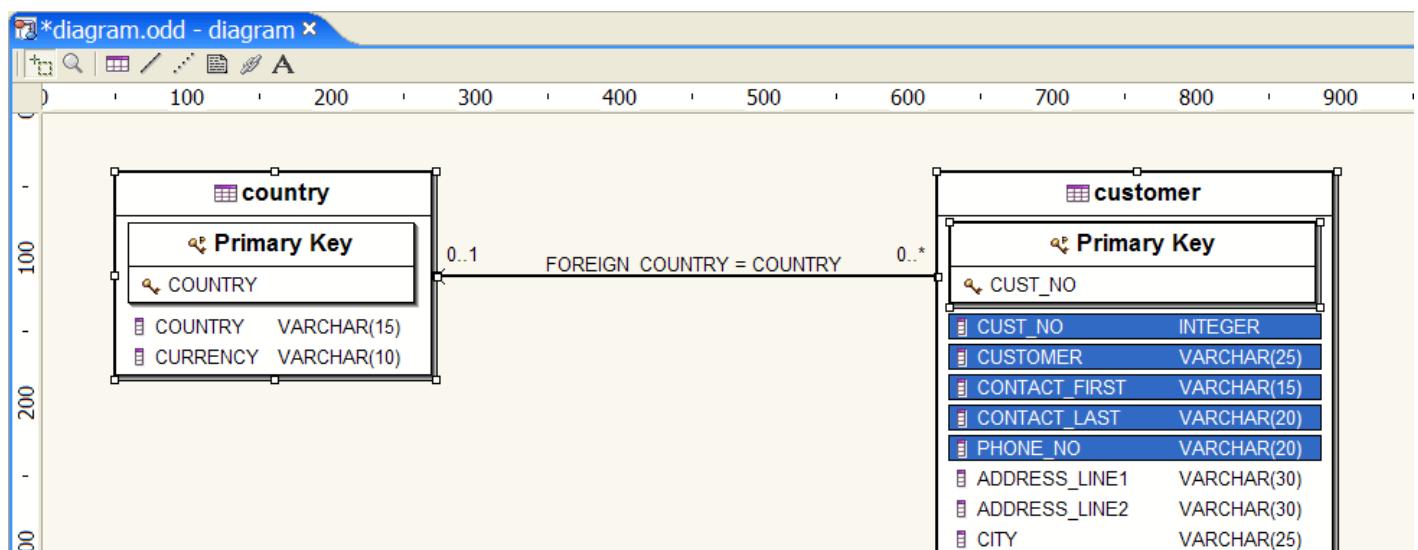
2.1. Selection mode



Select the **Selection mode** tool.

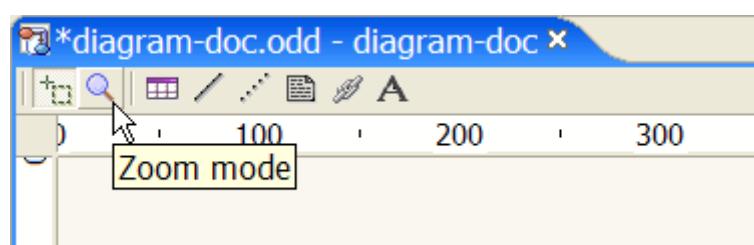


Drag a set of Database Objects.

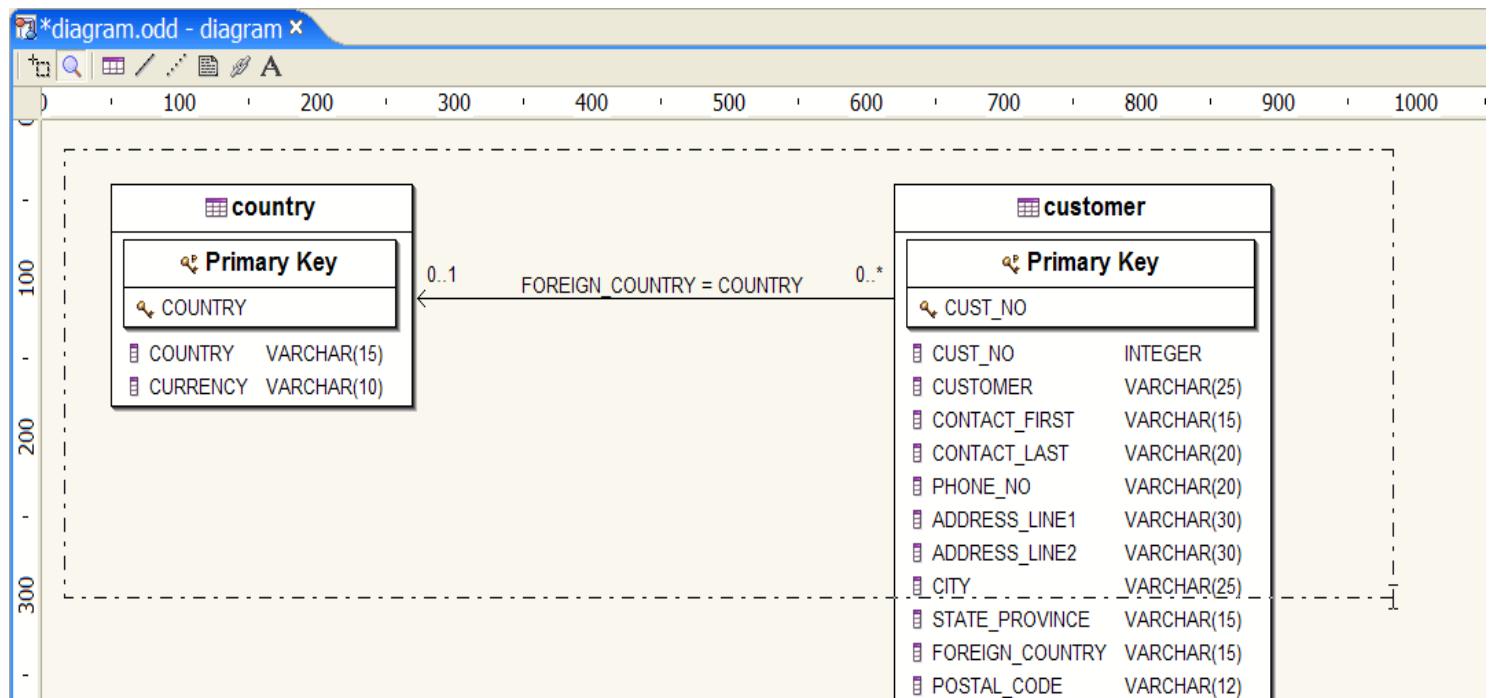


As a result, a set of Database Objects is selected.

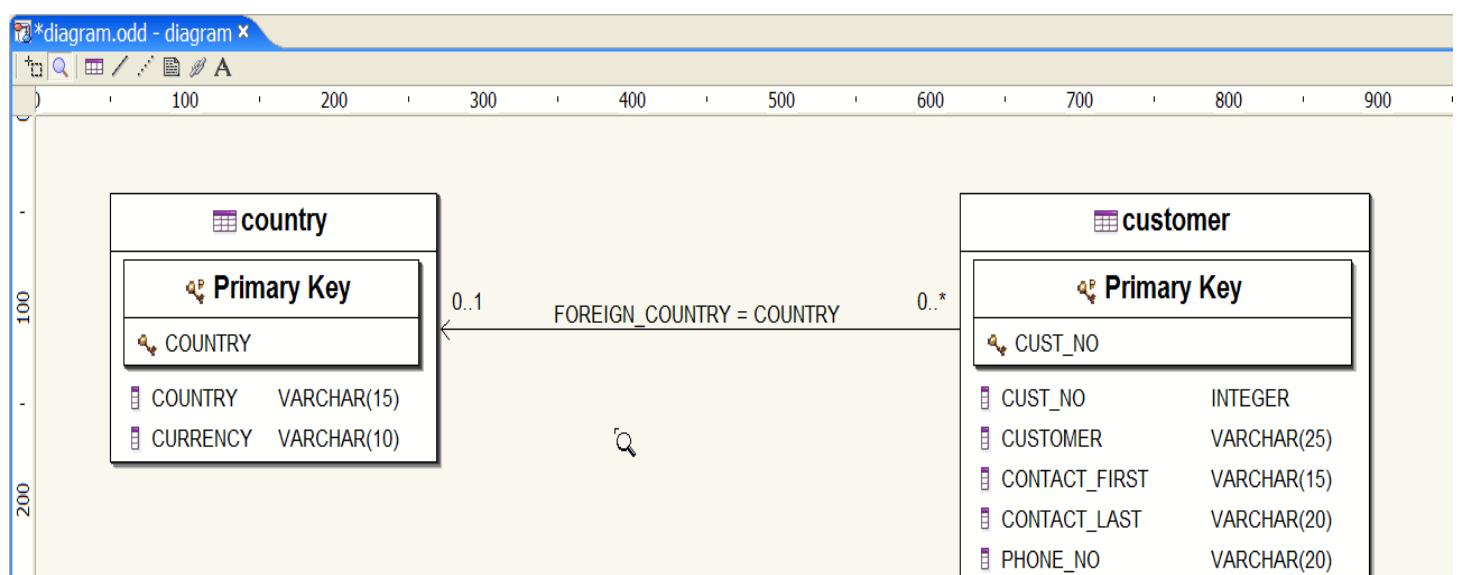
2.2. Zoom mode



Select the **Zoom mode** tool.

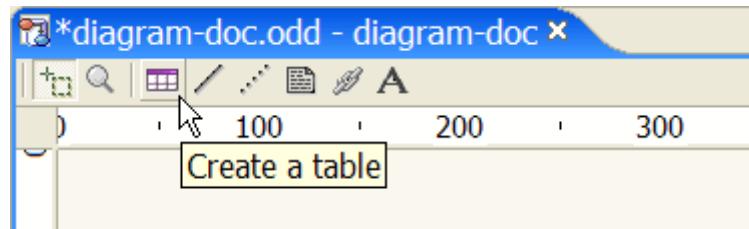


Drag a set of Database Objects.

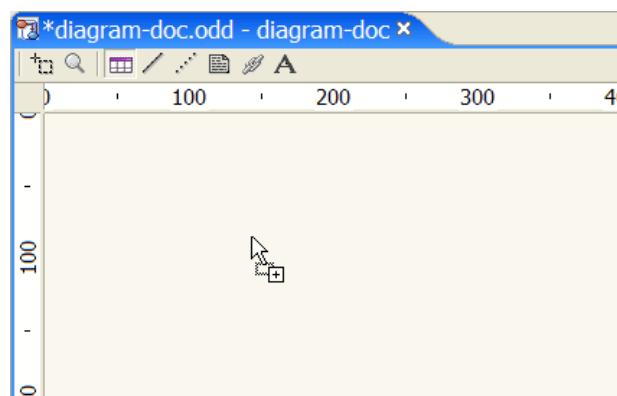


As a result, a set of Database Objects is zoomed.

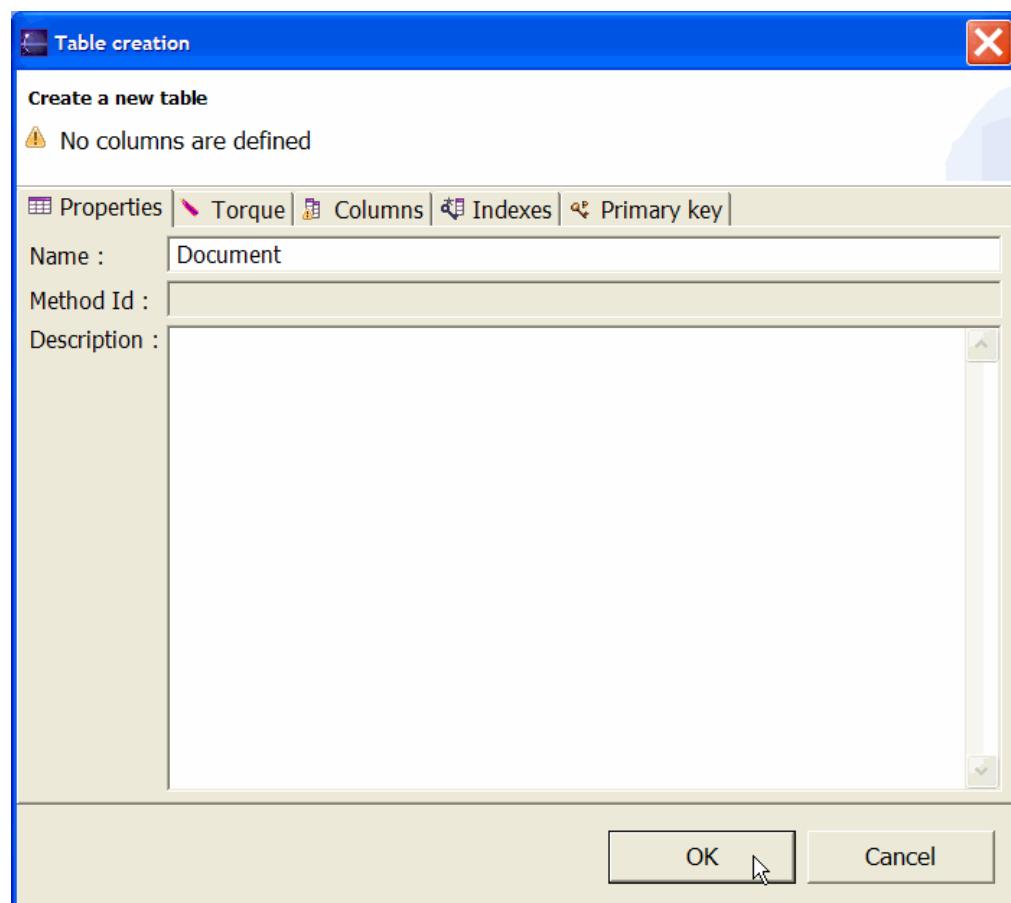
2.3. Create a table



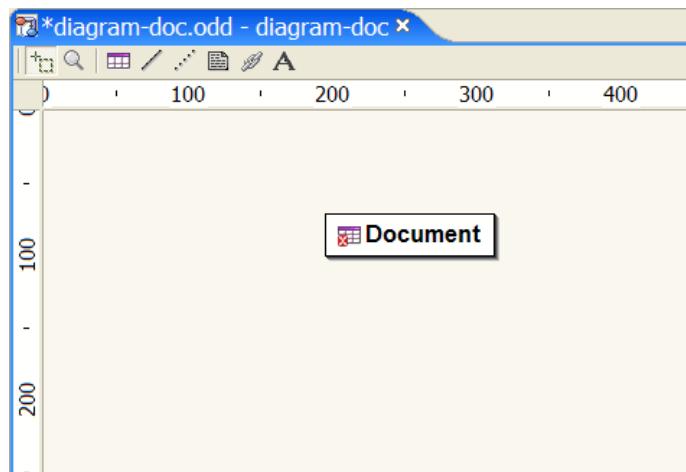
Select the **Create a table** tool.



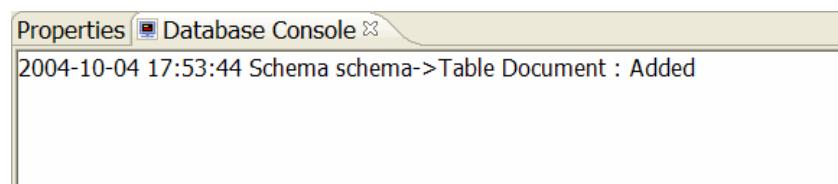
Target an area in your diagram or size your new Database Table.



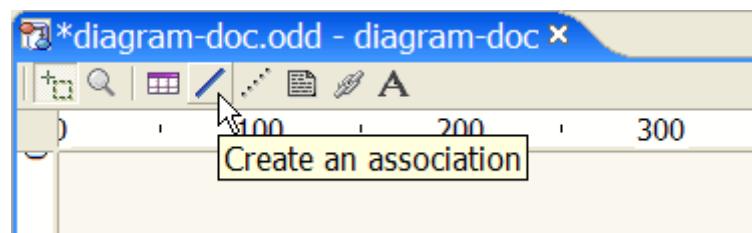
As a result, a Database Table is created.



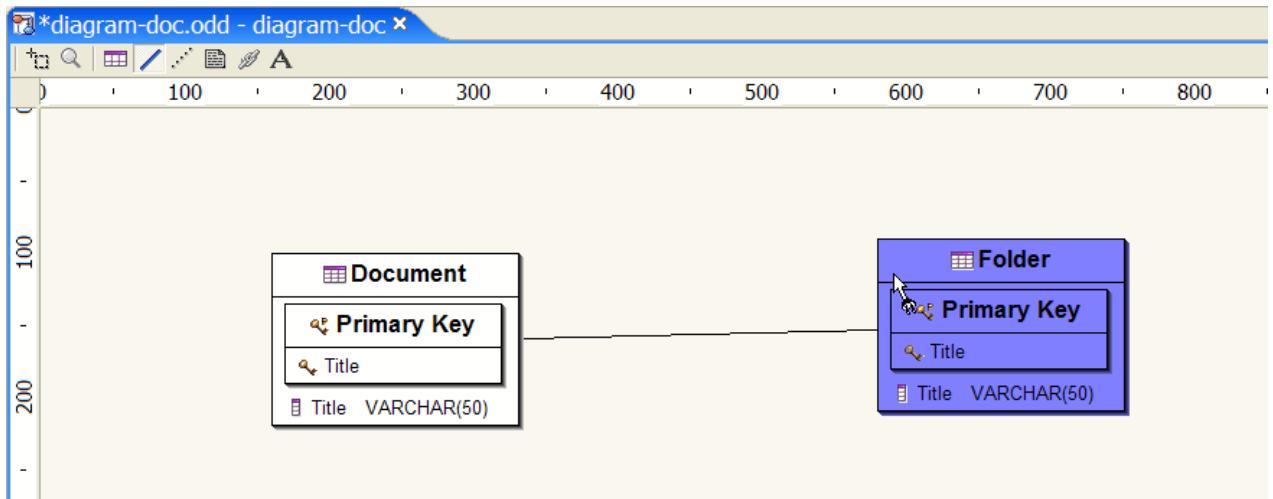
This Database Table has an error as no Database Columns have been defined.



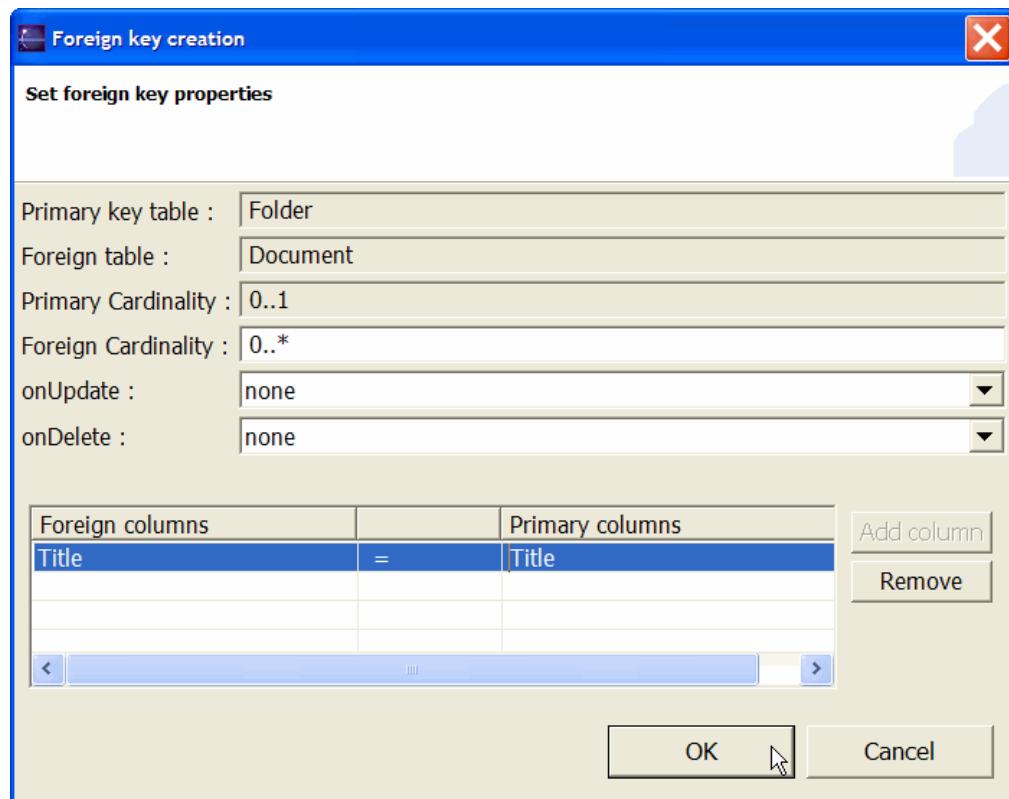
2.4. Create an association



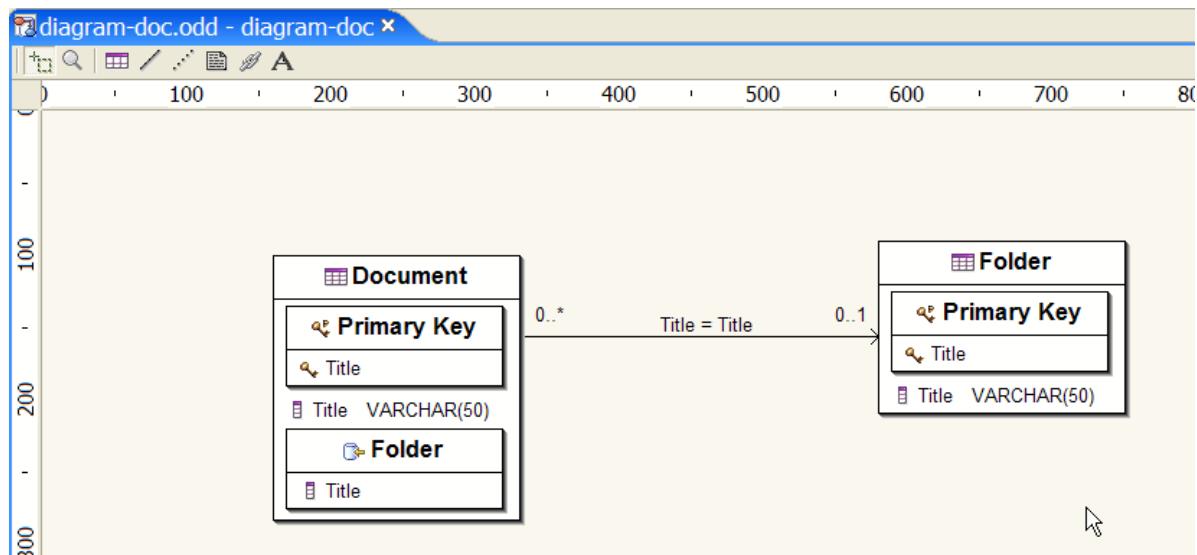
Select the **Create an association** tool.



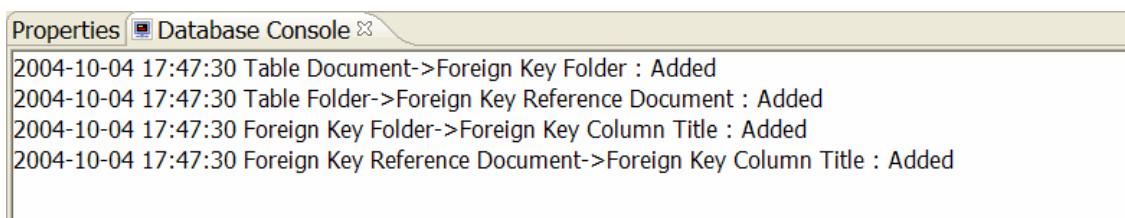
Select a Database Table and drag to the targeted Database Table.



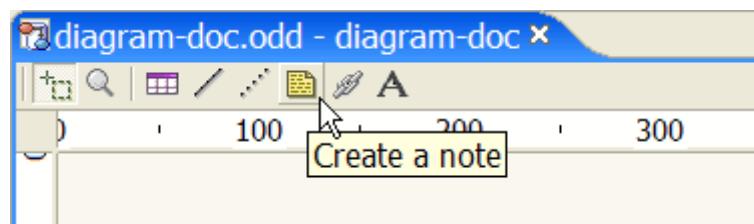
As a result an association is created.



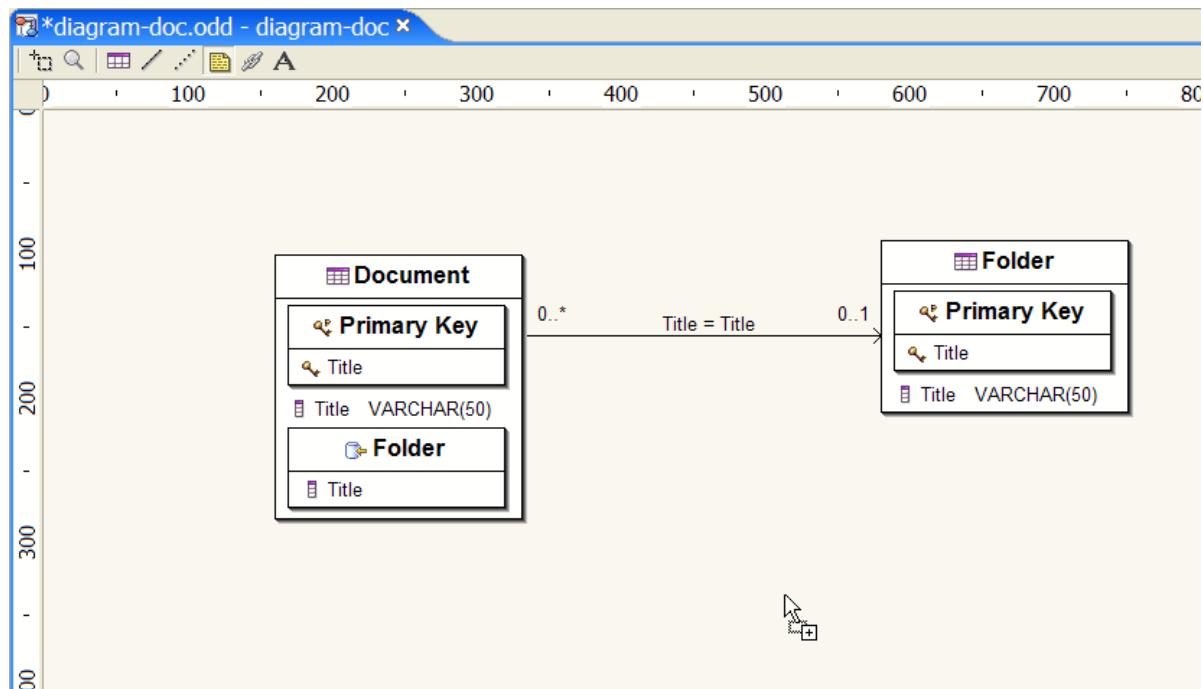
A detailed output is displayed in the [DatabaseConsole](#).



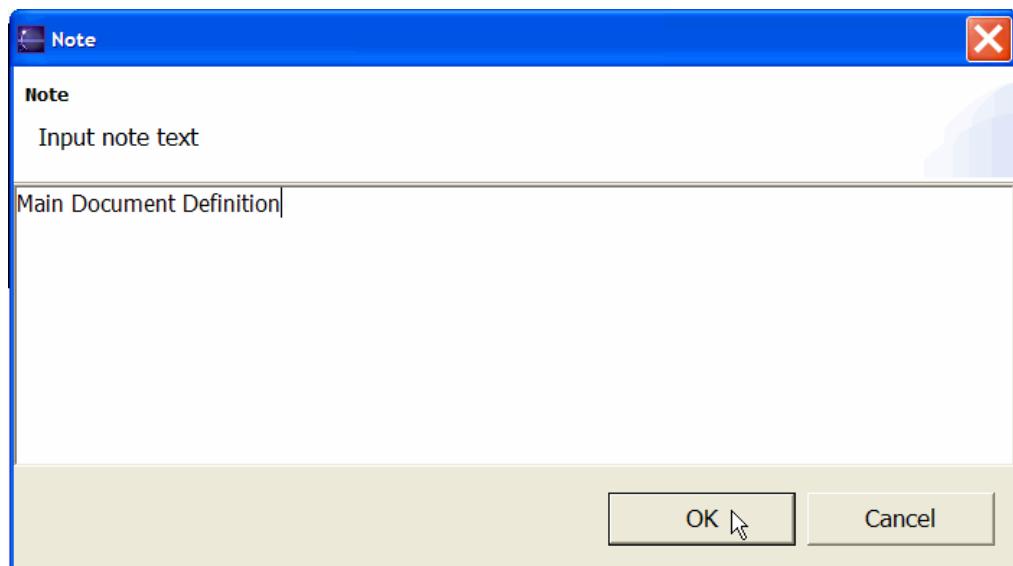
2.5. Create a note



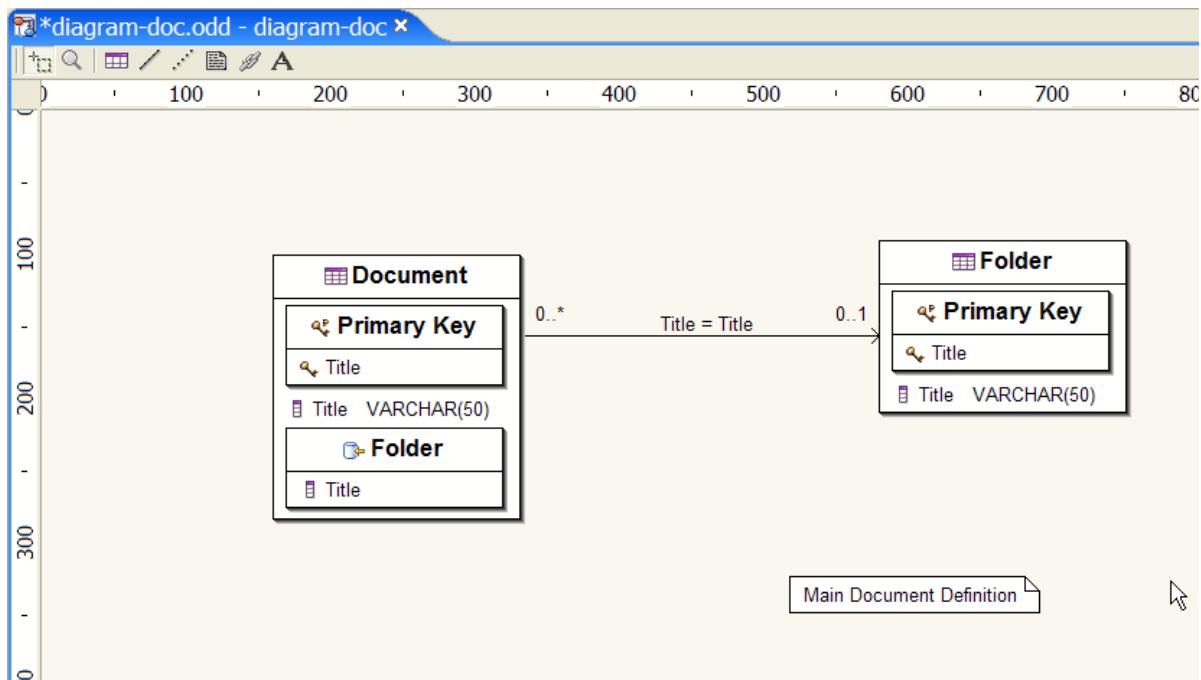
Select the **Create a note** tool.



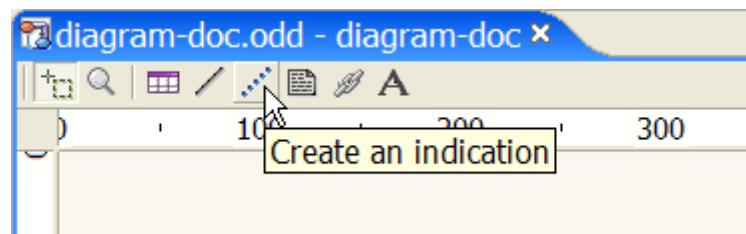
Target an area in your diagram or size your new Note.



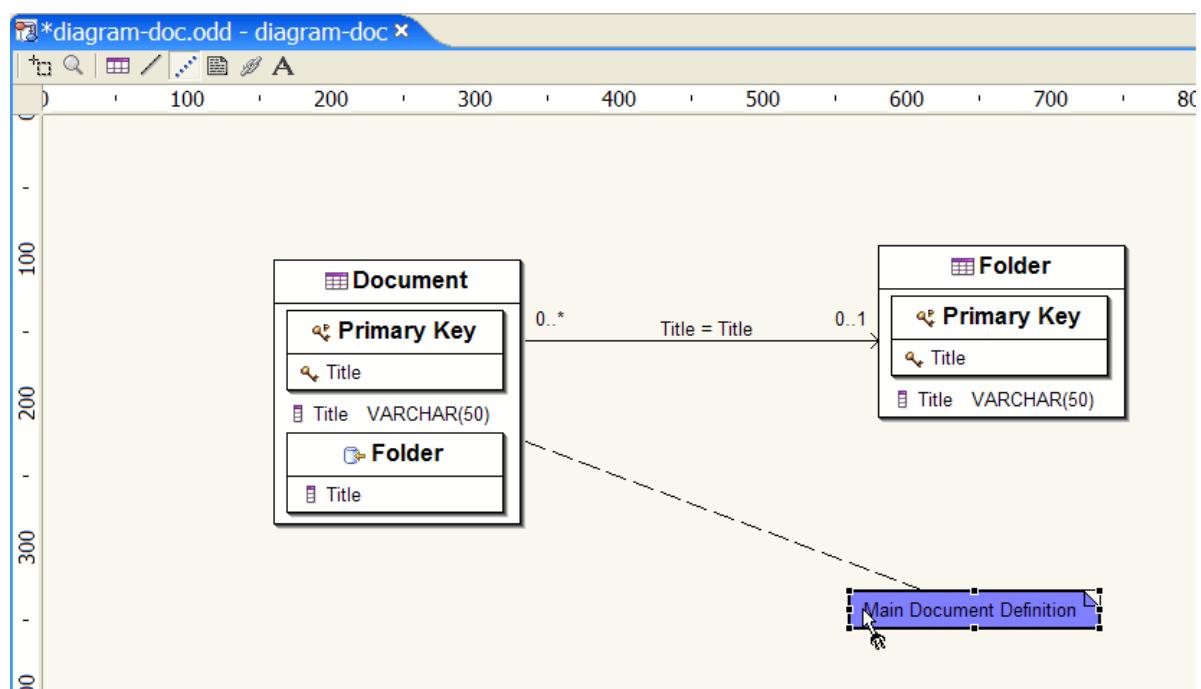
As a result a new Note is created.



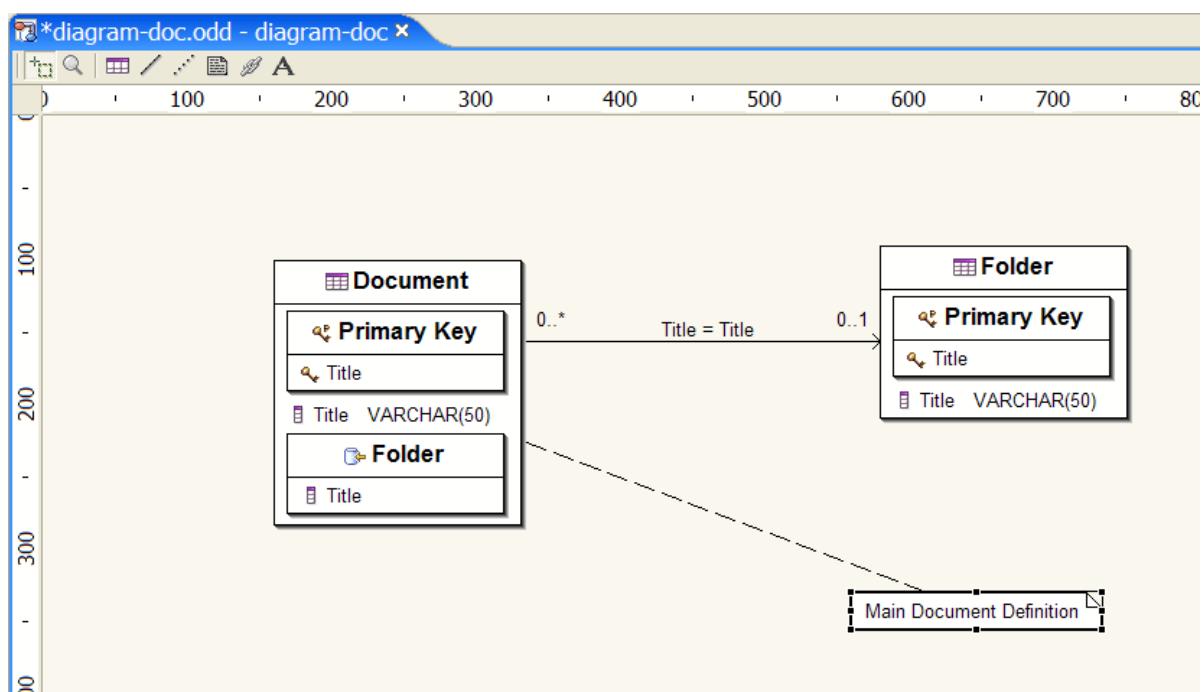
2.6. Create an indication



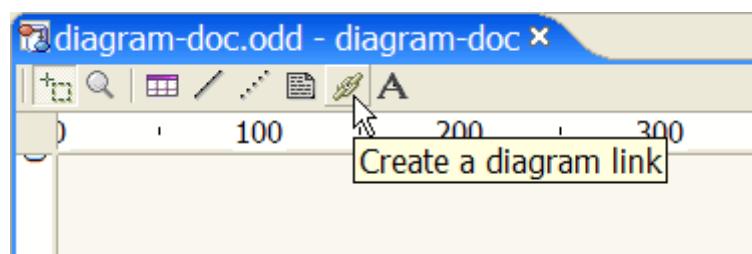
Select the **Create an indication** tool.



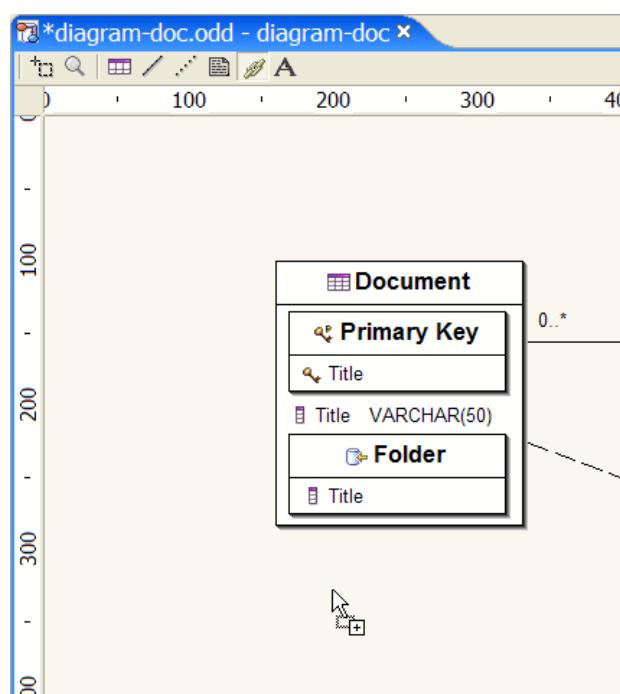
Select a Database Object and drag to the targeted Note.



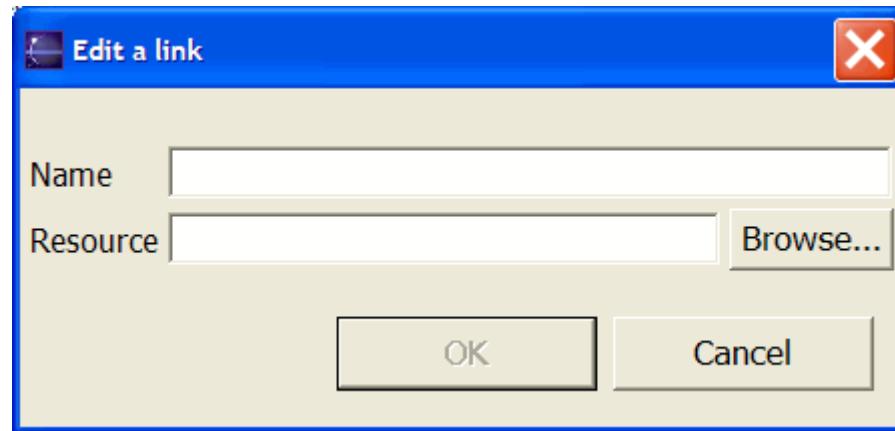
2.7. Create a diagram link



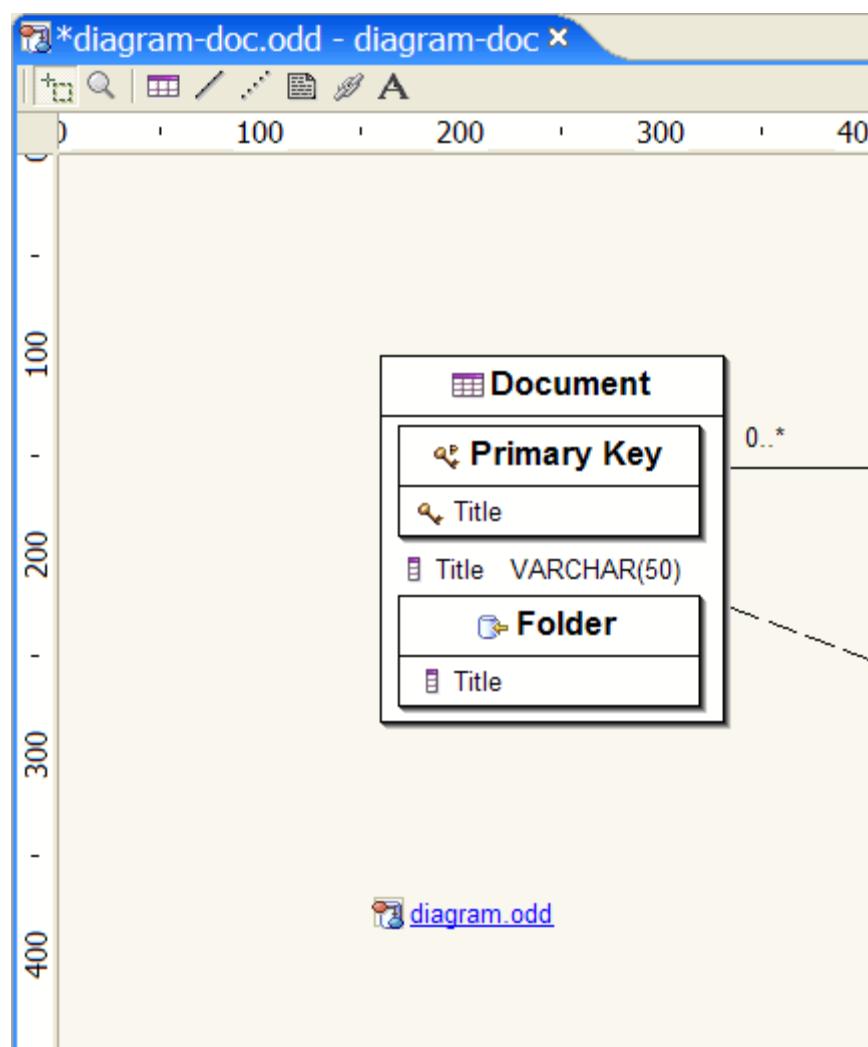
Select the **Create a diagram link** tool.



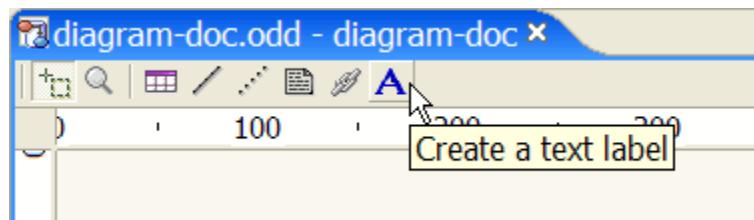
Target an area in your diagram or size your new link.



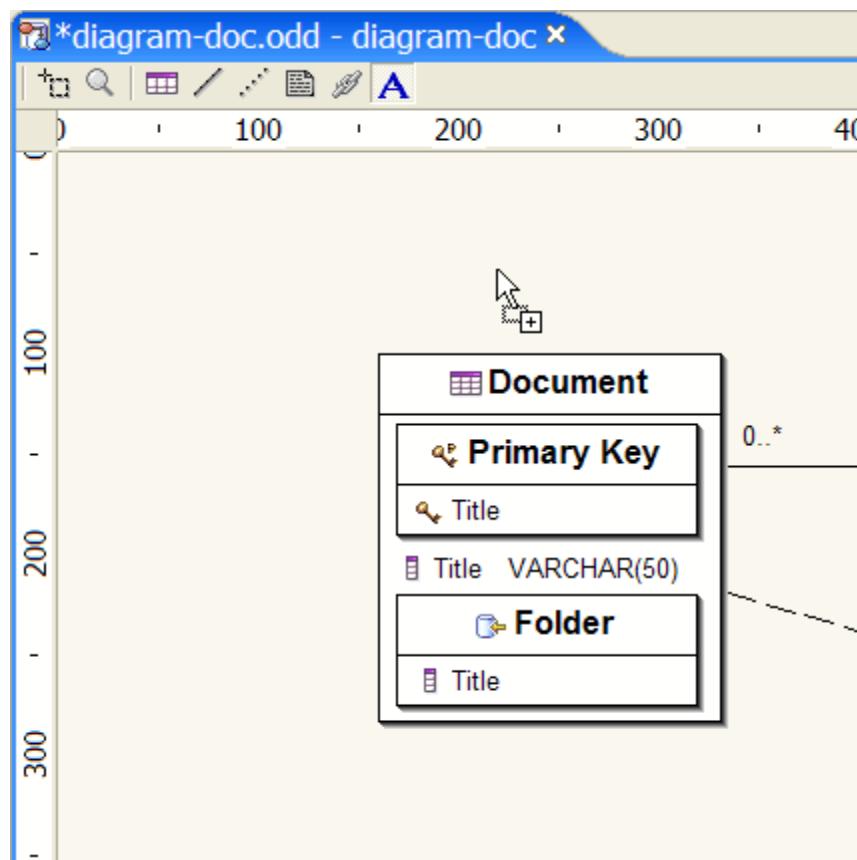
As a result a new link is created.



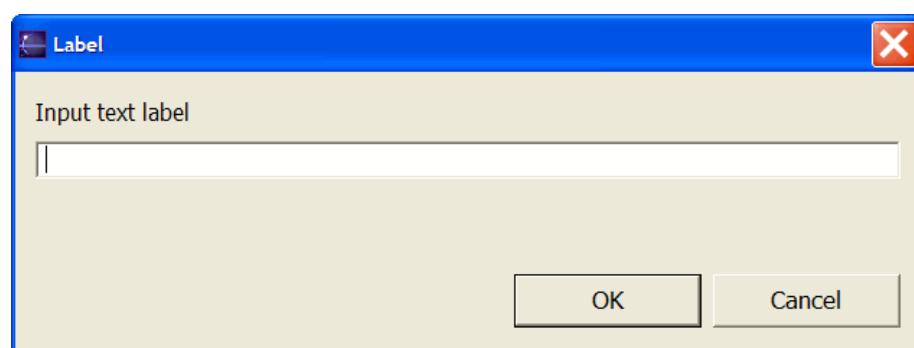
2.8. Create a text label



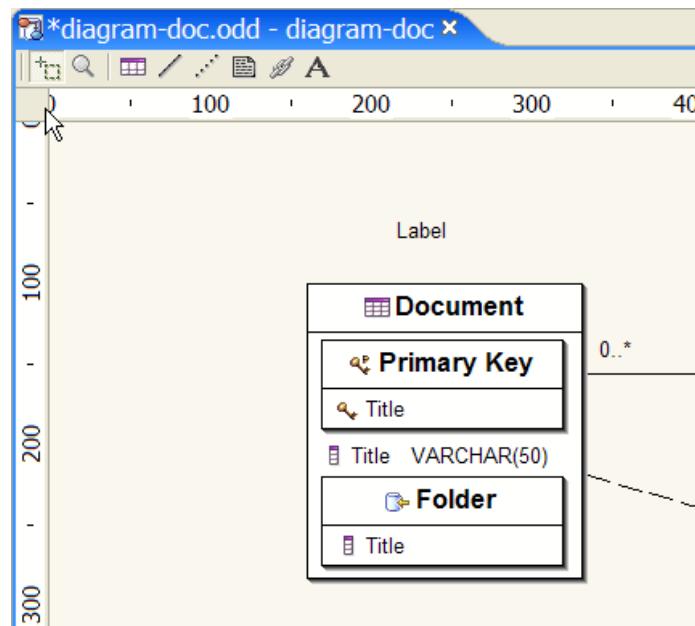
Select the **Create a text label** tool.



Target an area in your diagram or size your text label.



As a result a new text label is created.



Database Diagram Schema Selector

1. [Introduction](#)
2. [Database Schema Selector](#)
 1. [Schema tables](#)
 2. [Select All](#)
 3. [Deselect All](#)

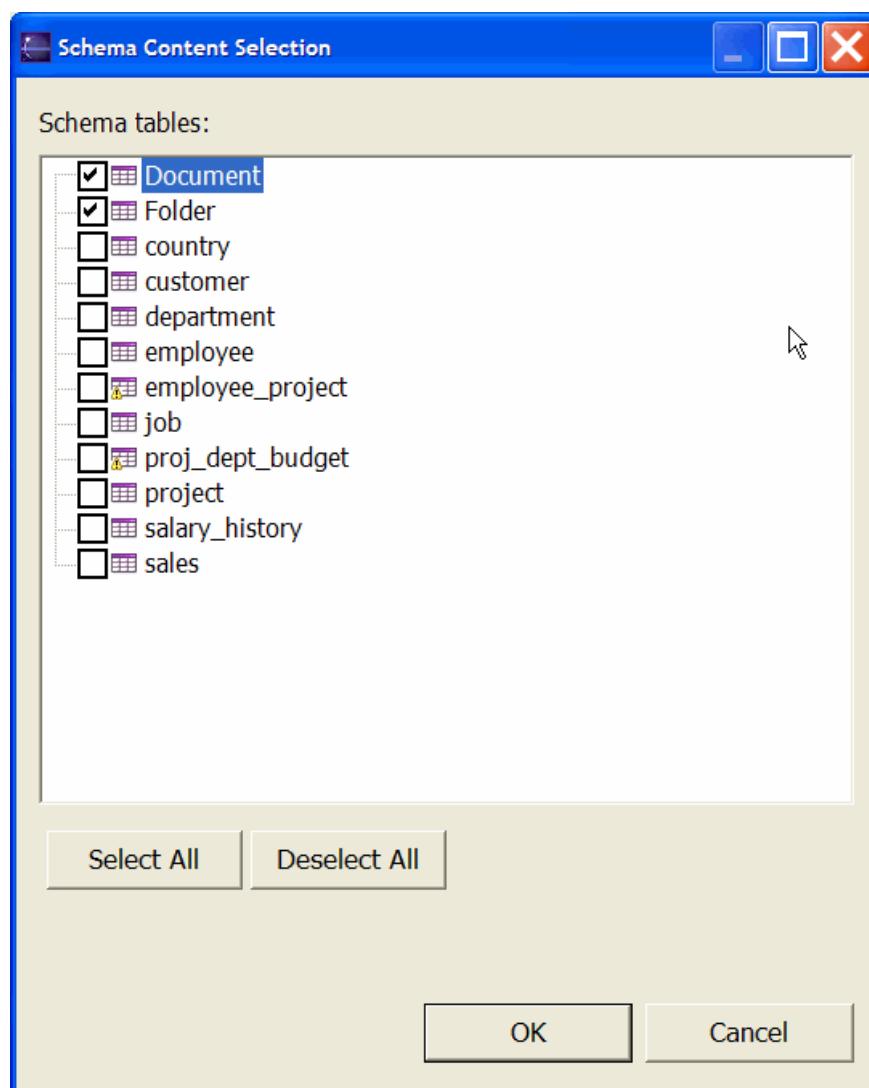
1. Introduction

In this section you will learn how to use the Database Schema Selector.

2. Database Schema Selector

Be sure that nothing is selected in the Diagram Editor, right-click and select :

Schema elements.



2.1. Schema tables

At any time you can select or deselect displayed Database Tables in your current Diagram.

Use the appropriate check-box to select or deselect a particular Database Table.

2.2. Select All

The **Select All** is a shortcut to select in a row all the available Database Tables.

2.3. Deselect All

The **Deselect All** is a shortcut to deselect in a row all the available Database Tables.

Editor Preferences

1. Introduction
 2. Diagram
 3. Table
 4. Primary Keys
 5. Indexes
 6. Foreign Keys

1. Introduction

In this section you will learn how to use Database Editor preferences.

Depending on its nature, an element can keep its preferences.

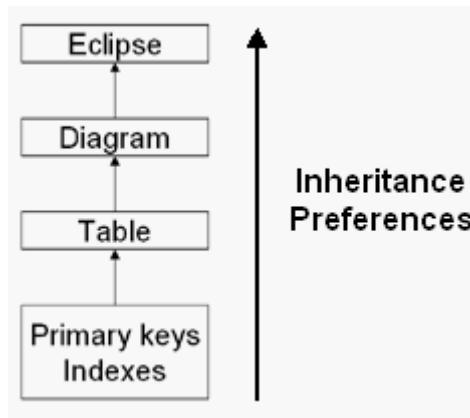
Every holder element of a diagram keeps its own preferences.

Changing [Database Diagram Global Preferences](#) will not impact your existing diagram's or element's preferences.

You can work on your Diagram to customize elements using the following :

1. Diagram
 2. Table
 3. Primary Keys
 4. Indexes
 5. Foreign Keys

Each element inherits the parent's value preferences or can have its own value.



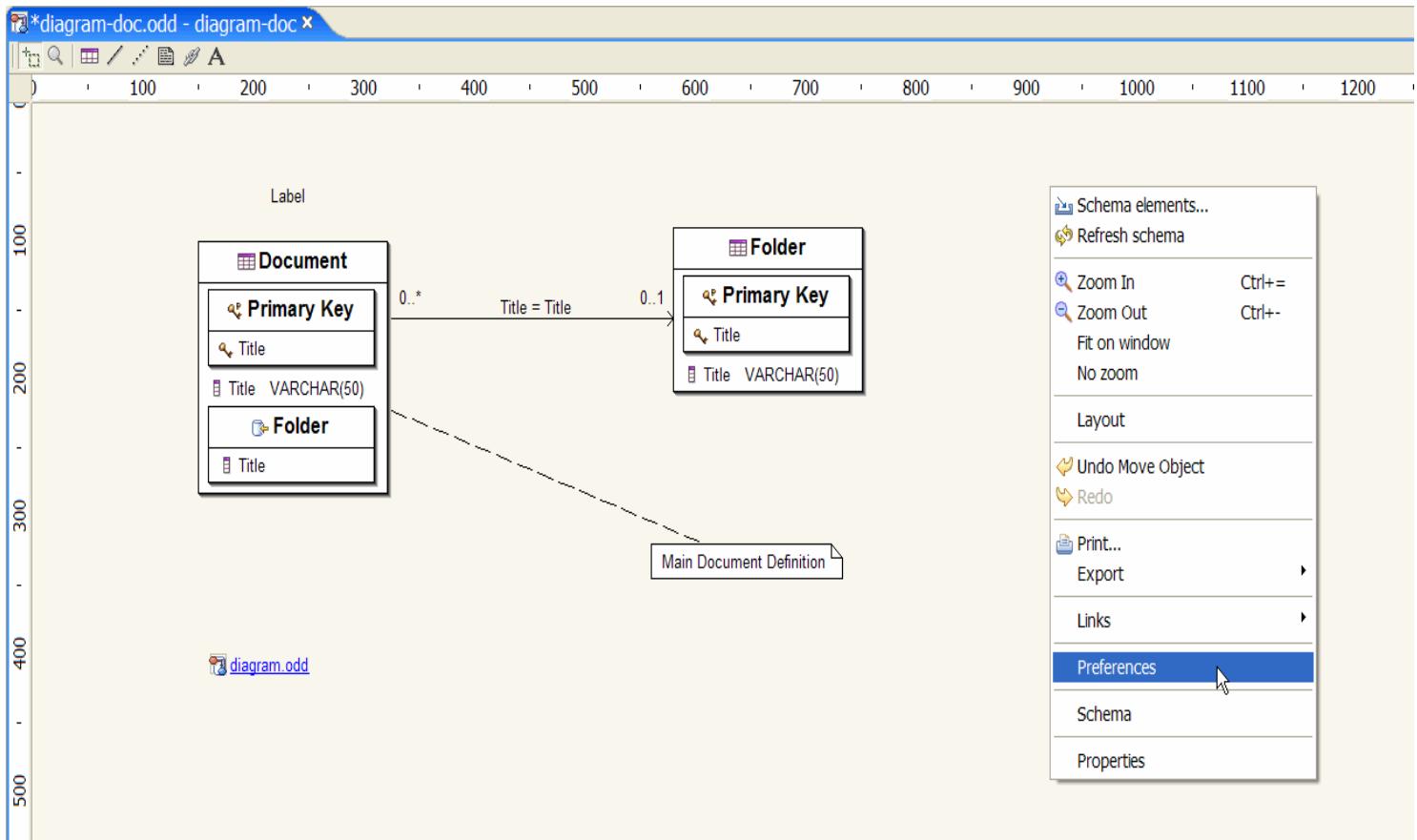
Diagram

1. Introduction
2. Diagram
 1. Appearance
 1. Packed view
 2. Free size
 3. Parent's value
 2. Columns
 1. Show
 2. Hide
 3. Parent's value
 3. Show all
 4. Hide all
 5. Use parent's value
 6. Reset

1. Introduction

In this section you will learn how to use Database Diagram Editor preferences.

2. Diagram



Be sure that nothing is selected in the Diagram Editor, right-click and select :

Preferences



Four Tabbed folders are available :

- Tables
- Indexes
- Primary keys
- Foreign Keys

2.1. Appearance

The * indicates the parent's value which has been selected in the [Database Diagram Global Preferences](#).

2.1.1. Packed view

Size is automatically managed according to its contents.

2.1.2. Free size

User managed size.

2.1.3. Parent's value

Inherited from the [Database Diagram Global Preferences](#).

2.2. Columns

The * indicates the parent's value which has been selected in the [Database Diagram Global Preferences](#).

2.2.1. Show

Columns are displayed in the current Database Diagram.

2.2.2. Hide

Columns are hidden in the current Database Diagram.

2.2.3. Parent's value

Inherited from the [Database Diagram Global Preferences](#).

2.3. Show all

This command is a shortcut to display all the elements of a Database Diagram.
The four tabbed folders **Show** commands are selected accordingly.

2.4. Hide all

This command is a shortcut to hide all the elements of a Database Diagram.
The four tabbed folders **Hide** commands are selected accordingly.

2.5. Use parent's value

This command is a shortcut to inherit the [Database Diagram Global Preferences](#) for all elements.
The four tabbed folders commands are selected accordingly.

2.6. Reset

This command is a shortcut to reset all your preferences to the last saved state.
The four tabbed folders commands are selected accordingly.

Table

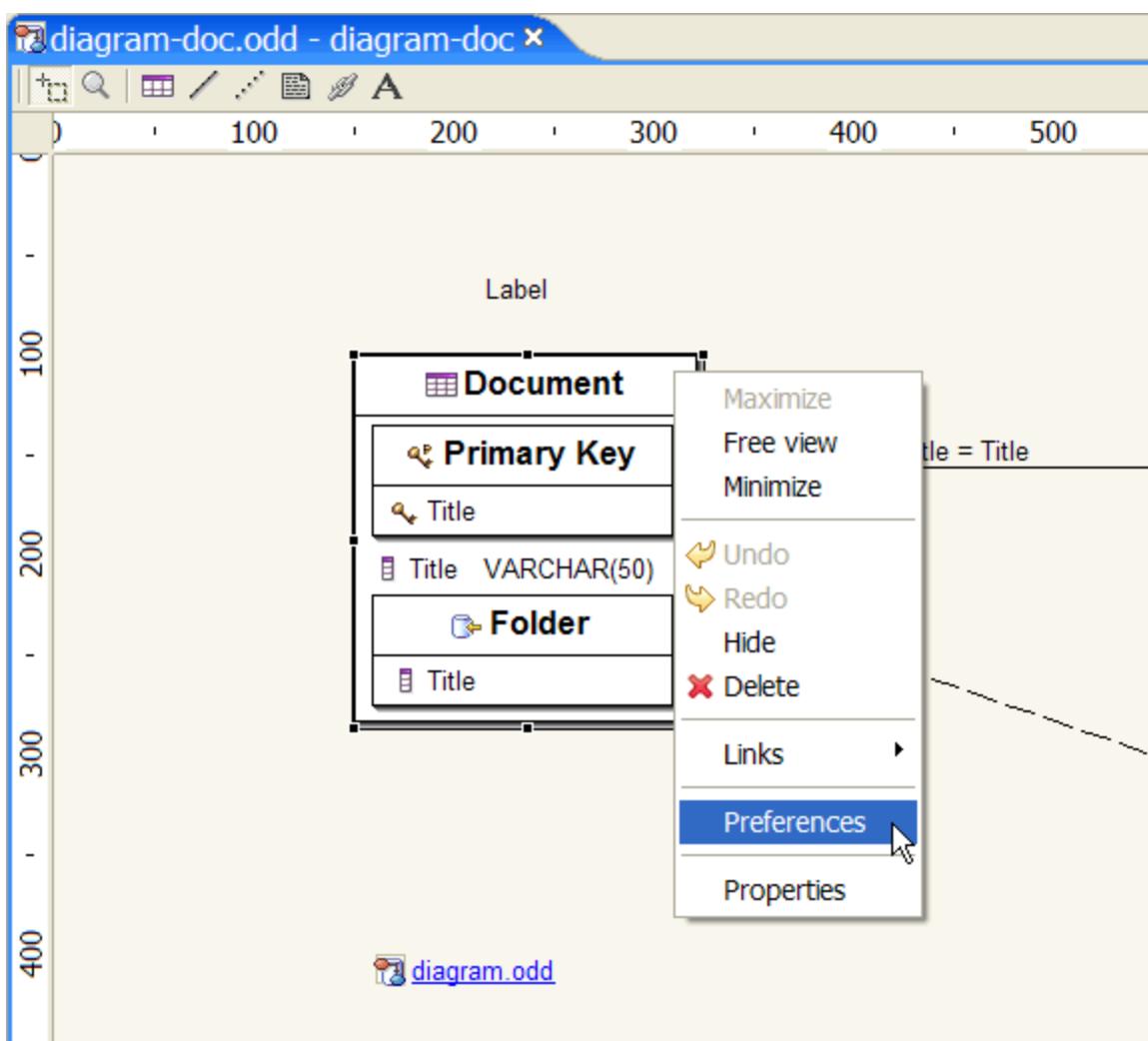
1. Introduction
2. Table
 1. Appearance
 1. Packed view
 2. Free size
 3. Parent's value
 2. Columns
 1. Show
 2. Hide
 3. Parent's value
 3. Show all
 4. Hide all
 5. Use parent's value
 6. Reset

1. Introduction

In this section you will learn how to use Database Editor Table preferences.

2. Table

You can customize your Database Diagram preferences.



Select a Database Table, right-click and select :
Preferences



Four Tabbed folders are available :

- Tables
- Indexes
- Primary keys
- Foreign Keys

2.1. Appearance

The * indicates the parent's value which has been selected in the [Database Diagram Preferences](#).

2.1.1. Packed view

Size is automatically managed according to its contents.

2.1.2. Free size

User managed size.

2.1.3. Parent's value

Inherited from the [Database Diagram Preferences](#).

2.2. Columns

The * indicates the parent's value which has been selected in the [Database Diagram Preferences](#).

2.2.1. Show

Columns are displayed in the current Database Diagram.

2.2.2. Hide

Columns are hidden in the current Database Diagram.

2.2.3. Parent's value

Inherited from the [Database Diagram Preferences](#).

2.3. Show all

This command is a shortcut to display all the elements of a Database Diagram.

The four tabbed folders **Show** commands are selected accordingly.

2.4. Hide all

This command is a shortcut to hide all the elements of a Database Diagram.
The four tabbed folders **Hide** commands are selected accordingly.

2.5. Use parent's value

This command is a shortcut to inherit the [Database Diagram Preferences](#) for all elements.
The four tabbed folders commands are selected accordingly.

2.2.6. Reset

This command is a shortcut to reset all your preferences to the last saved state.
The four tabbed folders commands are selected accordingly.

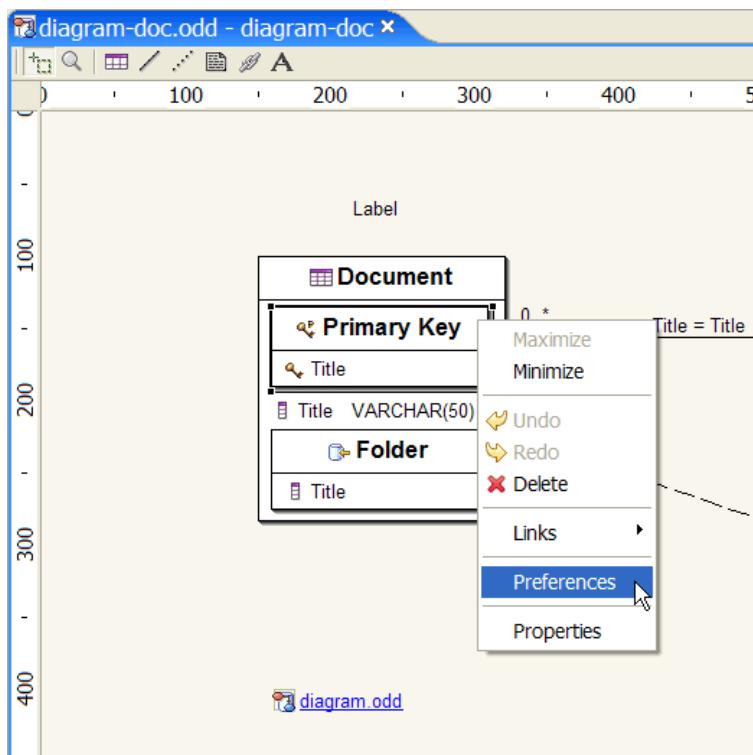
Primary Key

1. Introduction
2. Primary Keys
 1. Columns
 1. Show
 2. Hide
 3. Parent's value

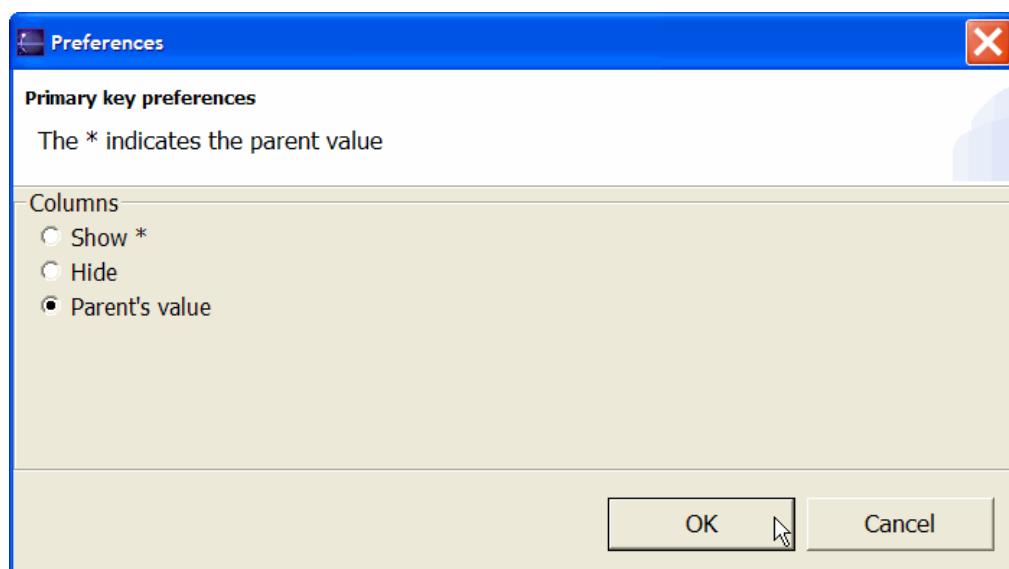
1. Introduction

In this section you will learn how to use Database Editor Primary Key preferences.

2. Primary Key



Select a Primary Key in a Database Table, right-click and select :
Preferences



2.1. Columns

The * indicates the parent's value which has been selected in the current [Table Preference](#).

2.1.1. Show

Columns are displayed in the current Primary Key.

2.1.2. Hide

Columns are hidden in the current Primary Key.

2.1.3. Parent's value

Inherited from the current [Table Preference](#).

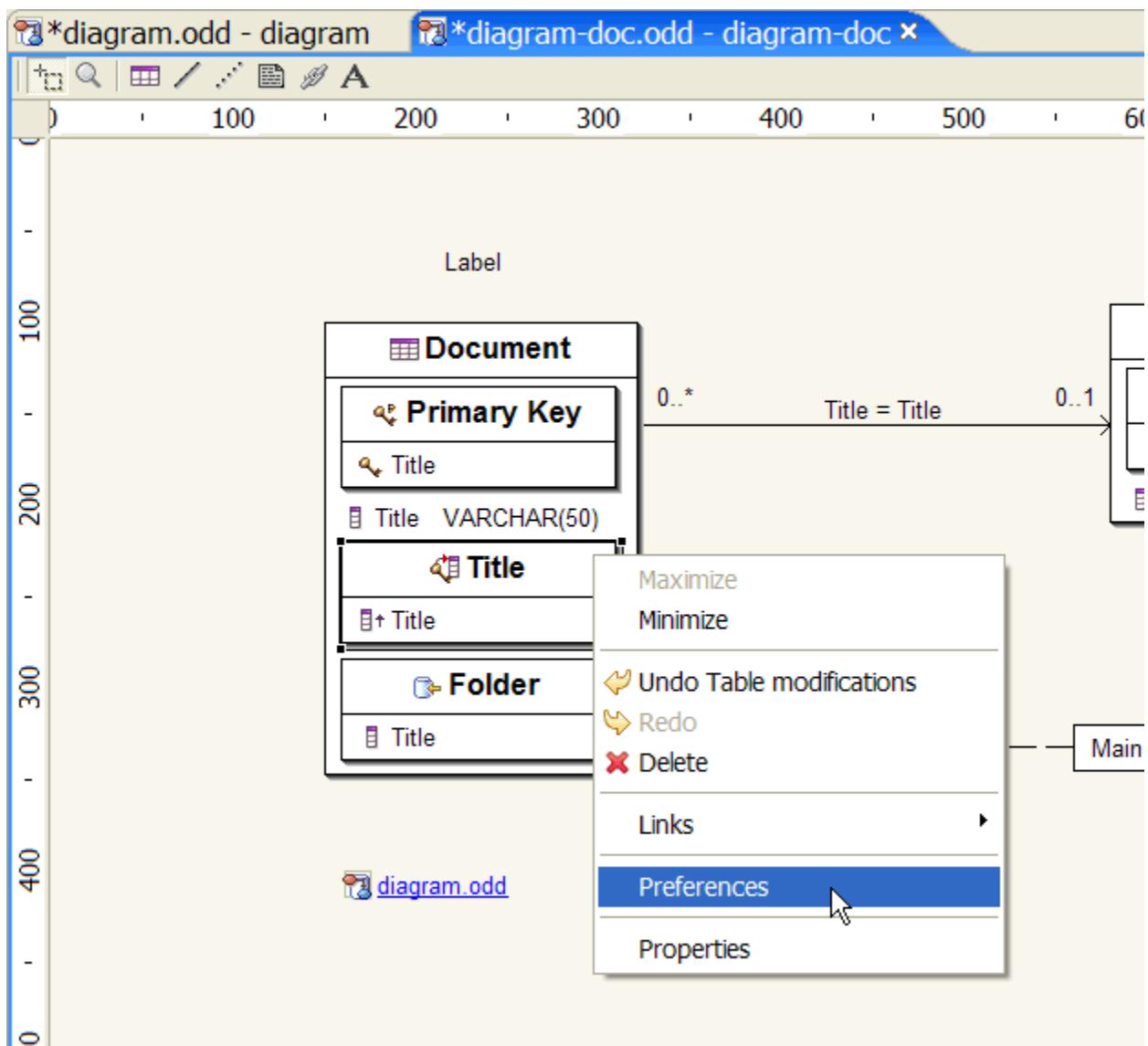
Index

1. Introduction
2. Index
 1. Columns
 1. Show
 2. Hide
 3. Parent's value

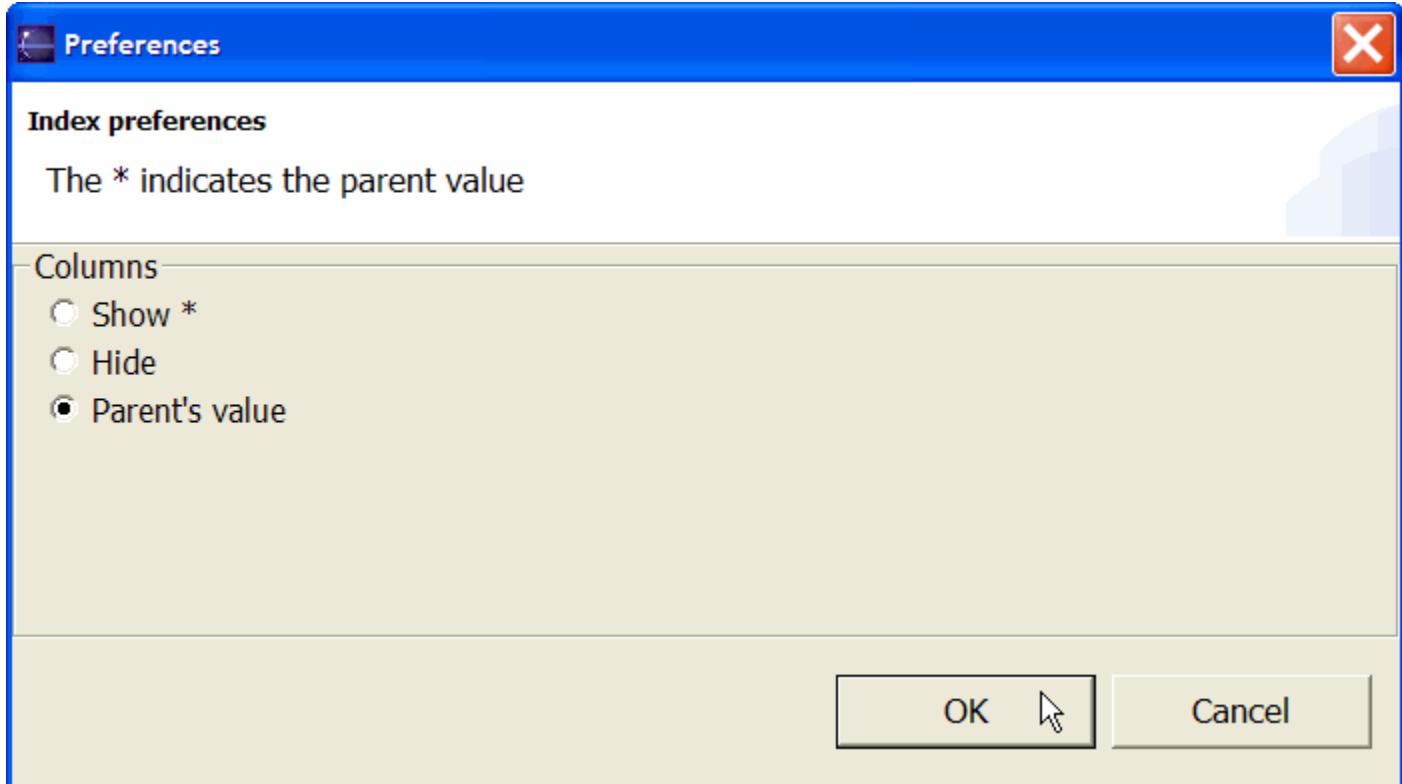
1. Introduction

In this section you will learn how to use Database Editor Index preferences.

2. Index



Select an Index in a Database Table, right-click and select :
Preferences



2.1. Columns

The * indicates the parent's value which has been selected in the current [Table Preference](#).

2.1.1. Show

Columns are displayed in the current Index.

2.1.2. Hide

Columns are hidden in the current Index.

2.1.3. Parent's value

Inherited from the current [Table Preferences](#).

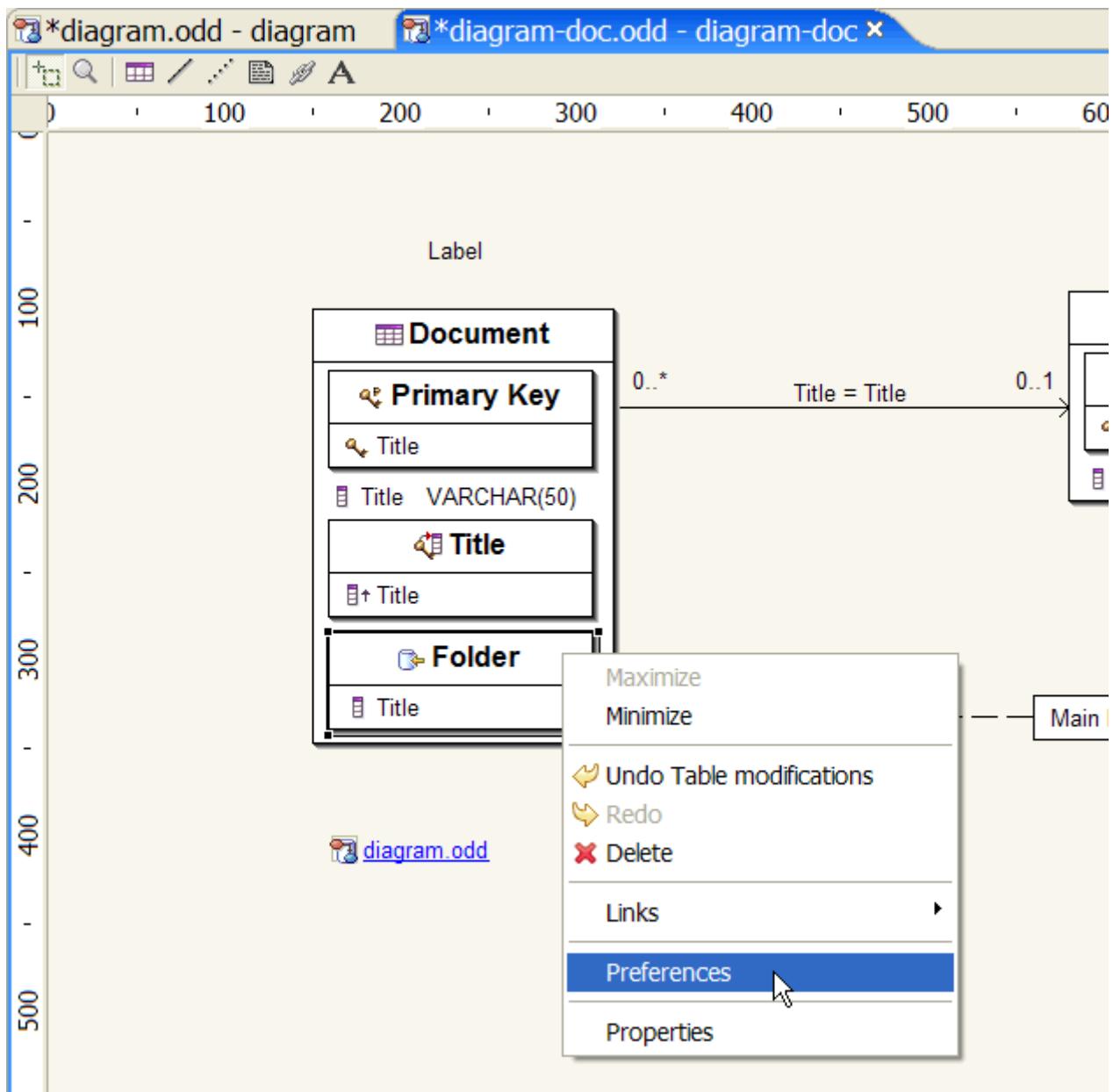
Foreign Key

1. Introduction
2. Foreign Key
 1. Columns
 1. Show
 2. Hide
 3. Parent's value

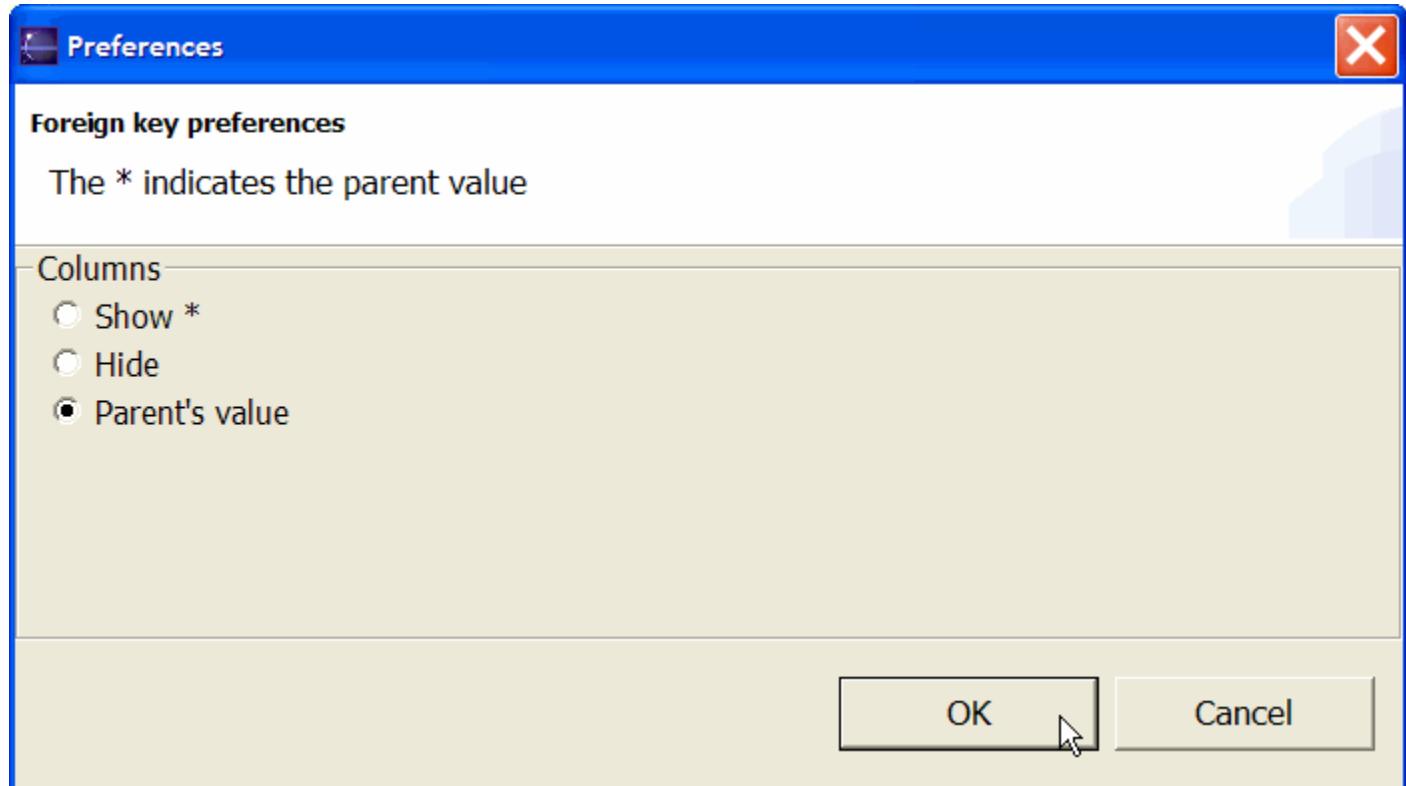
1. Introduction

In this section you will learn how to use Database Editor Foreign Key preferences.

2. Foreign Key



Select a Foreign Key in a Database Table, right-click and select :
Preferences



2.1. Columns

The * indicates the parent's value which has been selected in the current [Table Preference](#).

2.1.1. Show

Columns are displayed in the current Foreign Key.

2.1.2. Hide

Columns are hidden in the current Foreign Key.

2.1.3. Parent's value

Inherited from the current [Table Preference](#).

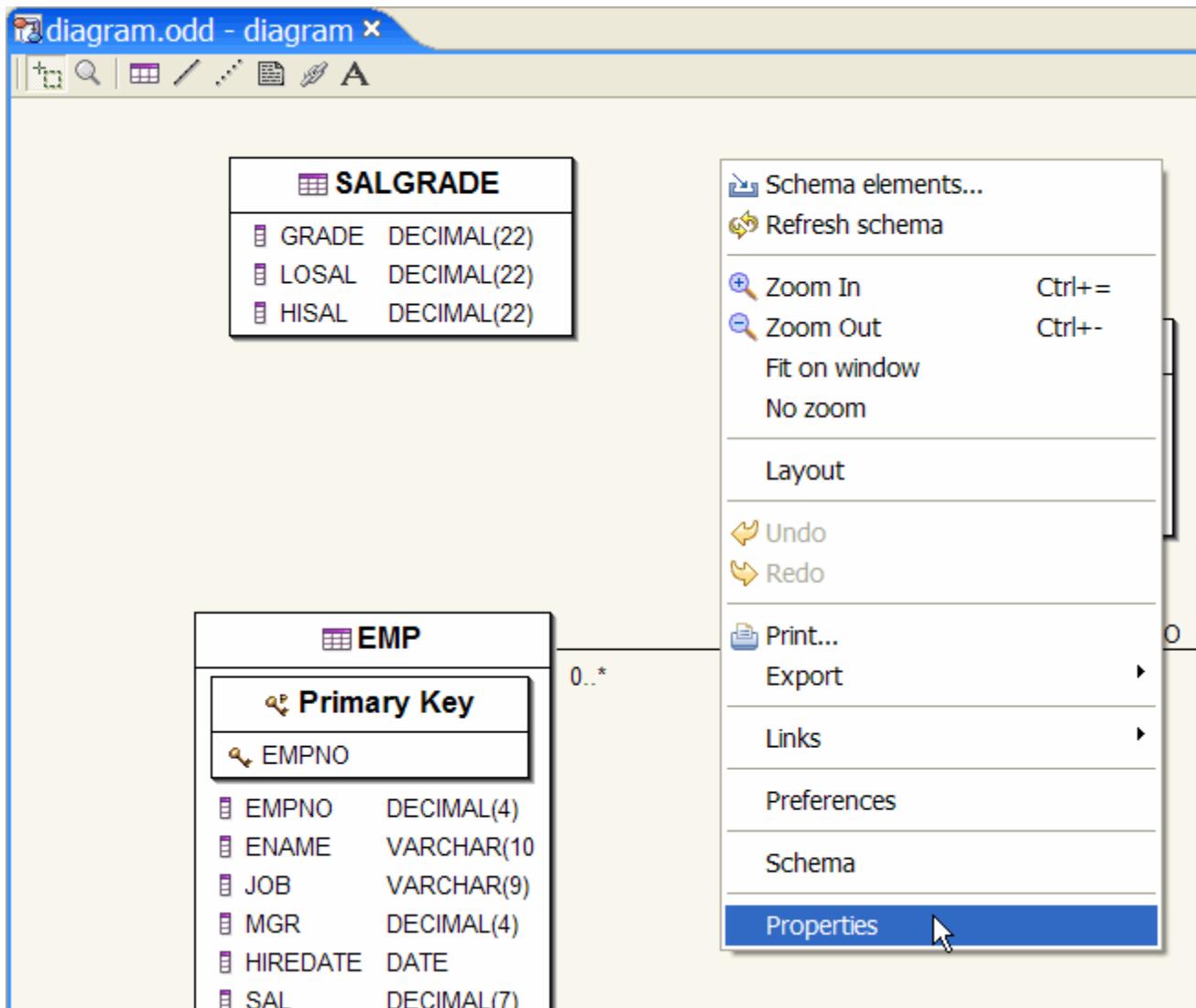
Database Editor Properties - Diagram

1. [Introduction](#)
2. [Diagram Board](#)

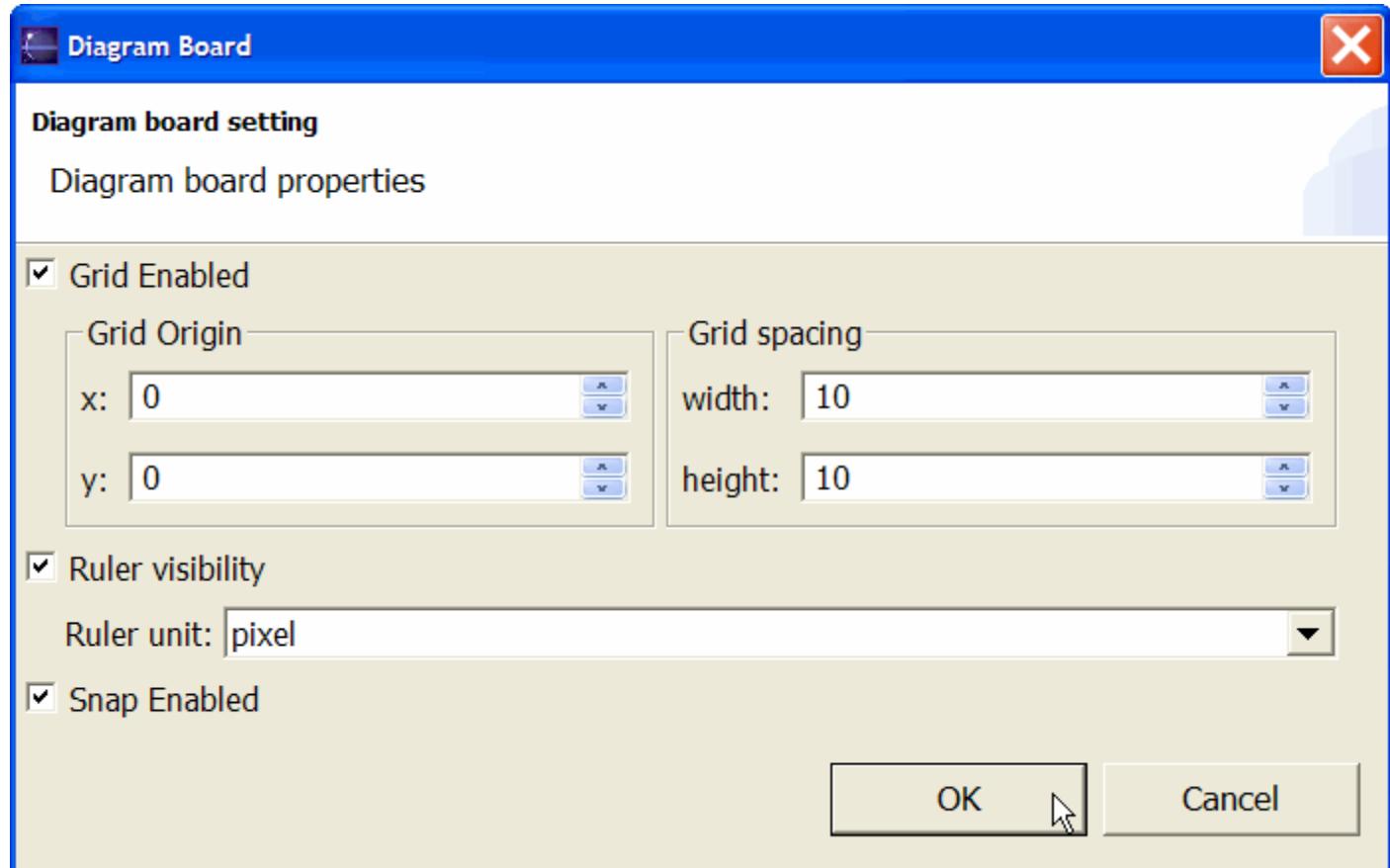
1. Introduction

In this section you will learn how to use Database Diagram Editor properties.

2. Diagram Board



Be sure that nothing is selected in the Diagram Editor, right-click and select : **Properties**



The Diagram Board page inherits the [UML Diagram Board Global Preferences](#) when you create a Database Diagram.

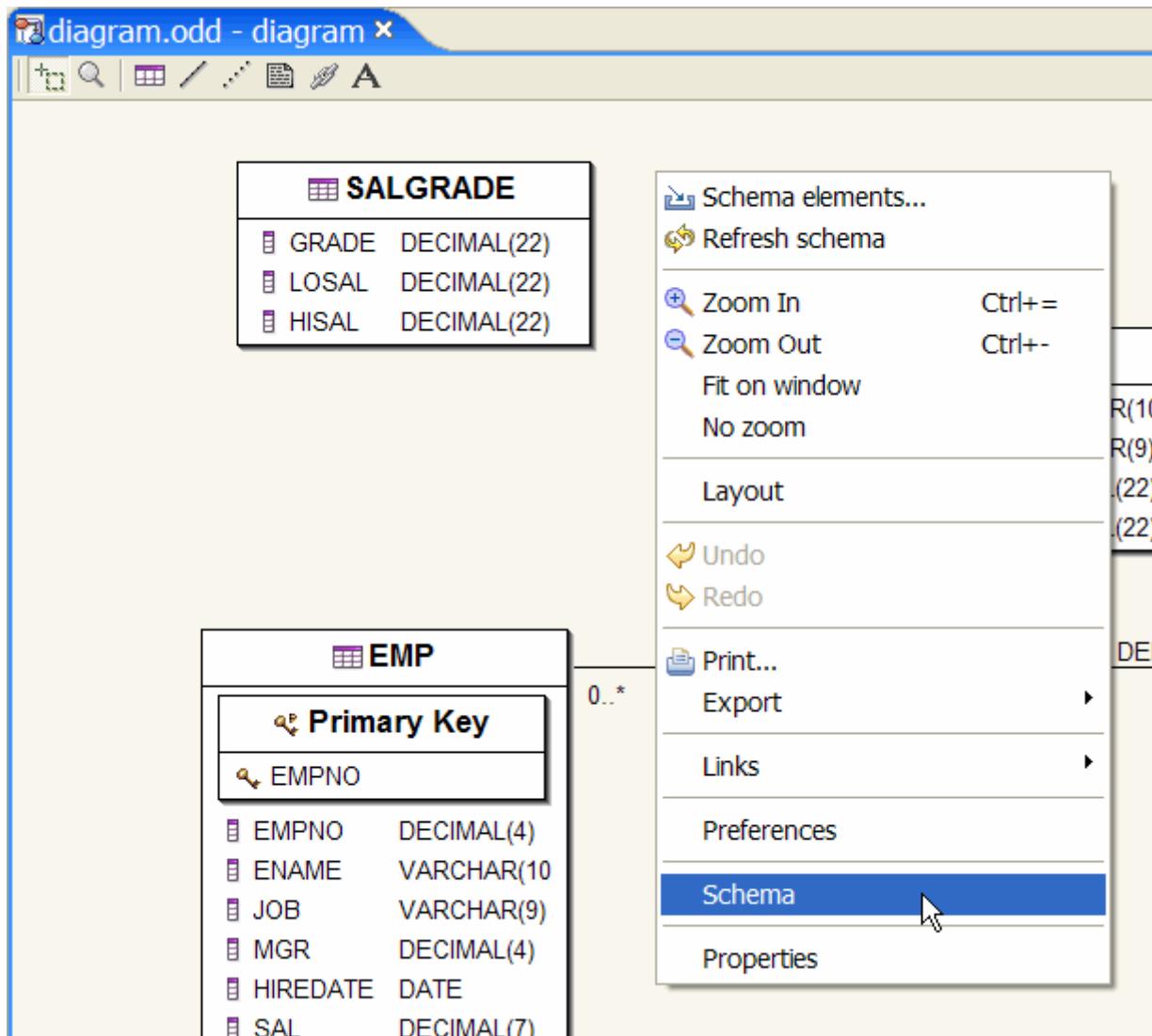
Database Editor Properties - Schema

1. Introduction
2. Schema Properties
 1. Schema
 2. Torque

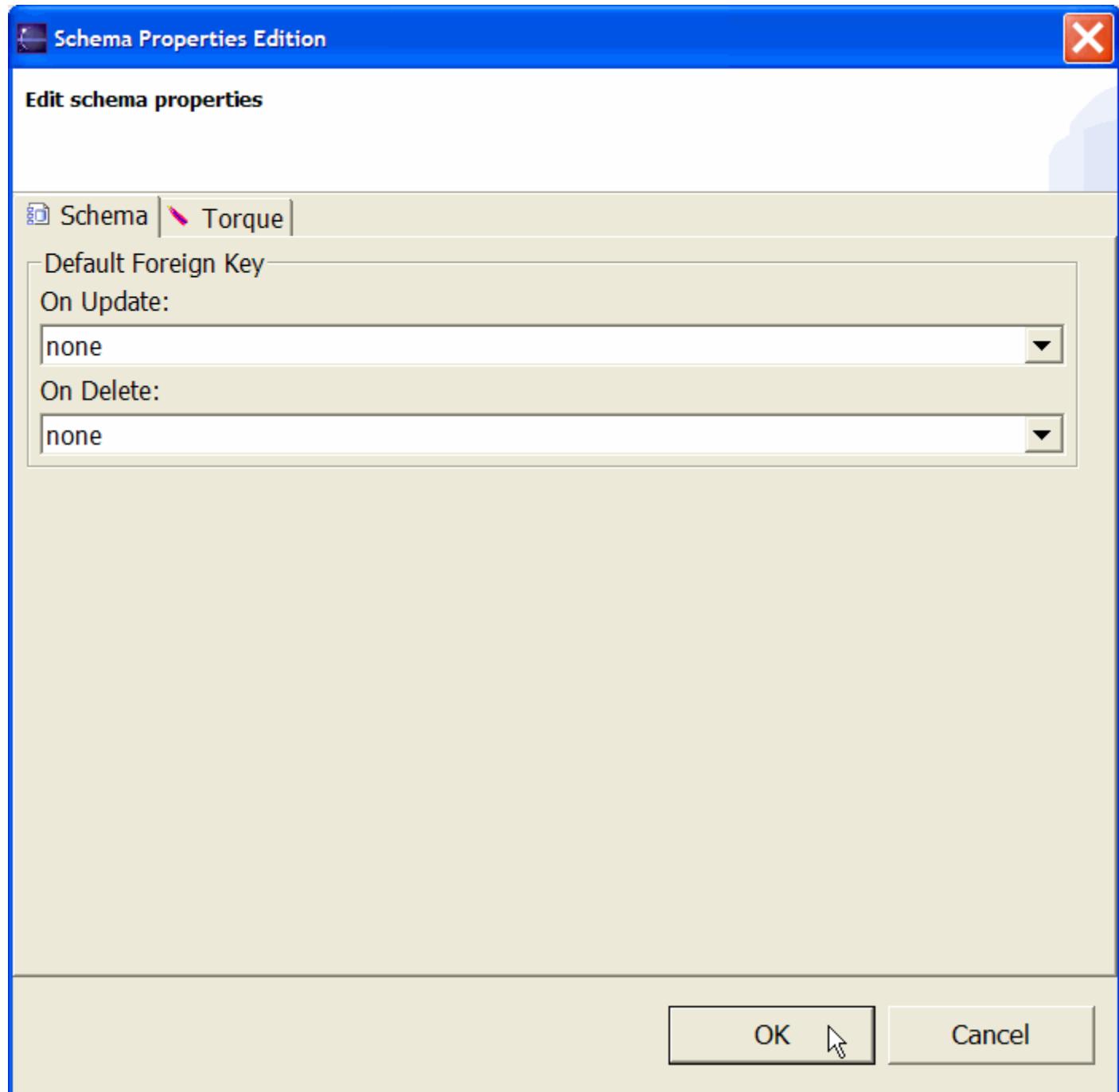
1. Introduction

In this section you will learn how to use Database Diagram Schema properties.

2. Schema Properties



Be sure that nothing is selected in the Diagram Editor, right-click and select : **Schema**



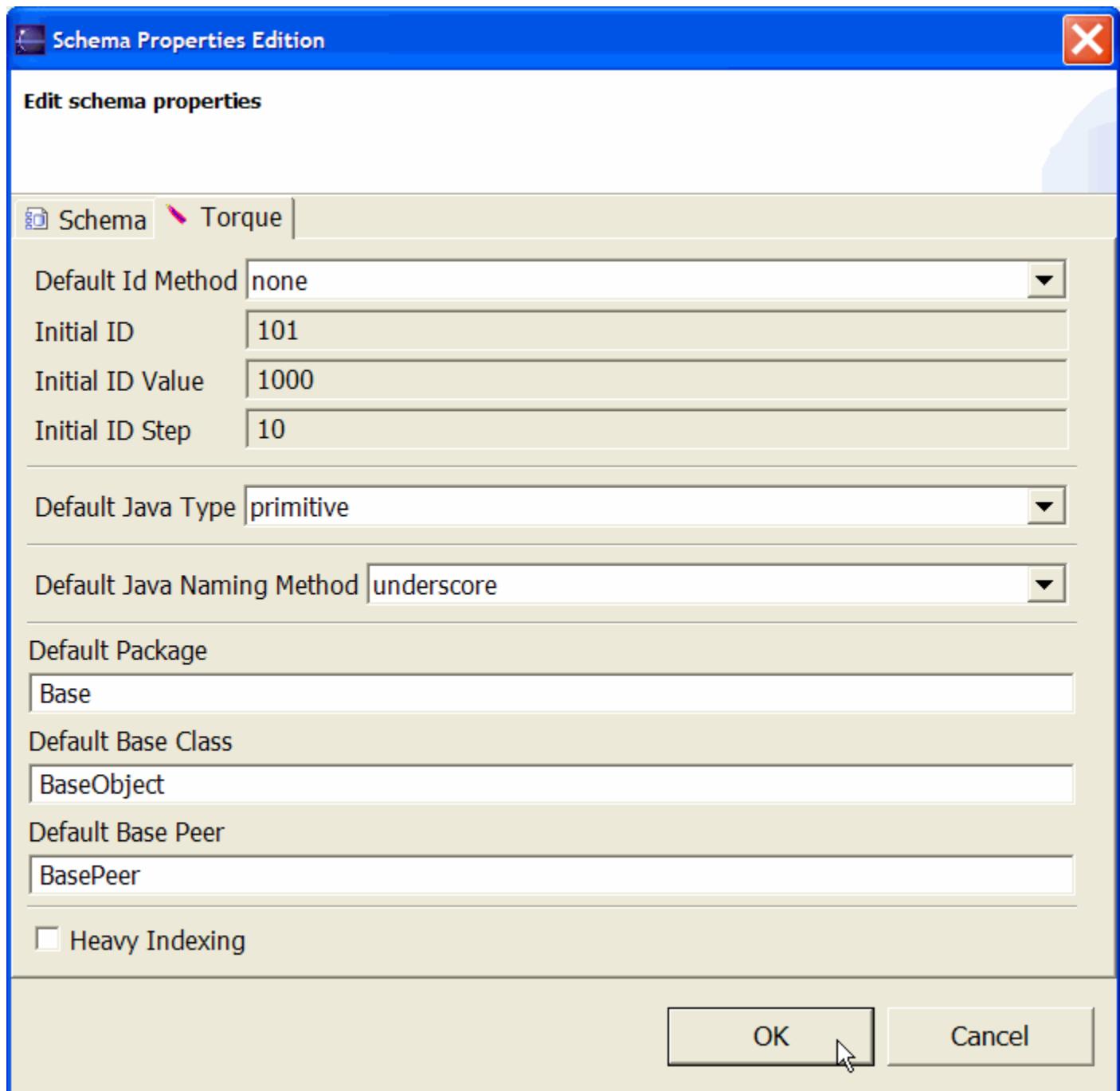
Two Tabbed folders are available :

- Schema
- Torque

2.1. Schema

The Database Schema page inherits the [Schema Default Foreign Key Global Preferences](#).

2.2. Torque



The Database Schema Torque page inherits the [Schema Torque Global Preferences](#). For further details about Torque options, the reader will find a detailed page [here](#).

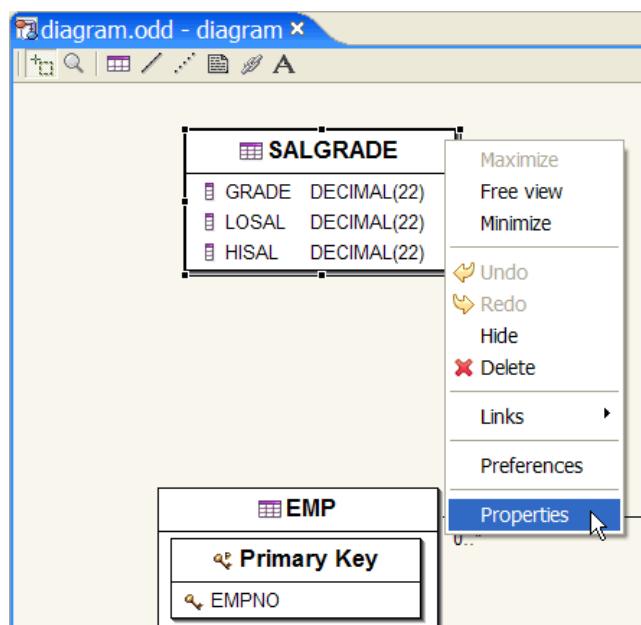
Database Editor Properties - Table

1. [Introduction](#)
2. [Table](#)
 1. [Properties](#)
 1. [Name](#)
 2. [Description](#)
 2. [Torque](#)
 3. [Columns](#)
 1. [New](#)
 2. [Edit](#)
 3. [Remove](#)
 4. [Move Up](#)
 5. [Move Down](#)
 4. [Indexes](#)
 1. [Indexes](#)
 1. [New](#)
 2. [Edit](#)
 3. [Remove](#)
 2. [Columns](#)
 1. [Add Column](#)
 2. [Remove](#)
 3. [Move Up](#)
 4. [Move Down](#)
 5. [Primary key](#)
 1. [Add Column](#)
 2. [Remove](#)
 3. [Move Up](#)
 4. [Move Down](#)
 6. [Foreign keys](#)
 1. [Foreign Keys](#)
 1. [New](#)
 2. [Remove](#)
 2. [Foreign Columns](#)
 1. [Add Column](#)
 2. [Remove](#)
 7. [OK](#)

1. Introduction

In this section you will learn how to use Database Editor Table properties.

2. Table

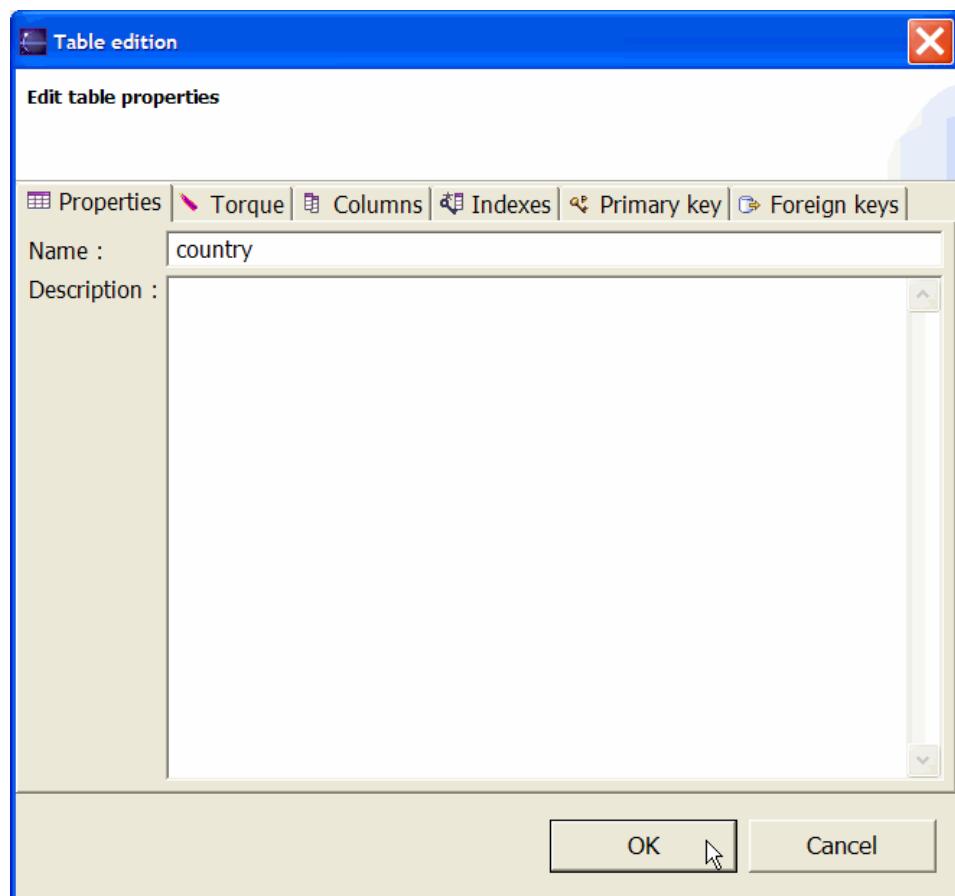


Select a Database Table, right-click and select :

Properties

or

Double Click a Database Table :



Six Tabbed folders are available :

- Properties
- Torque
- Columns
- Indexes
- Primary key
- Foreign keys

2.1. Properties

Table properties.

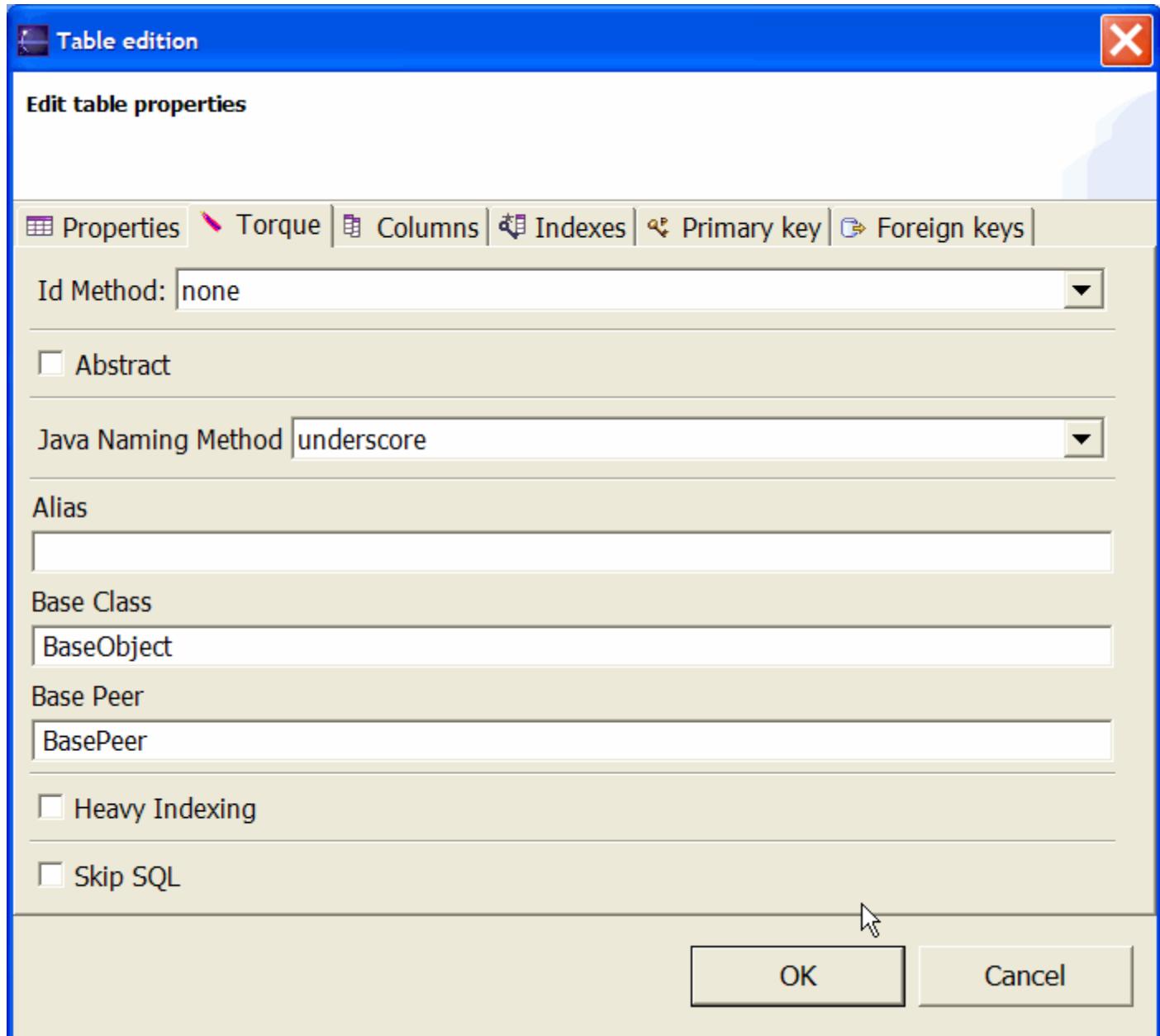
2.1.1. Name

Table name.

2.1.2. Description

Table description.

2.2. Torque

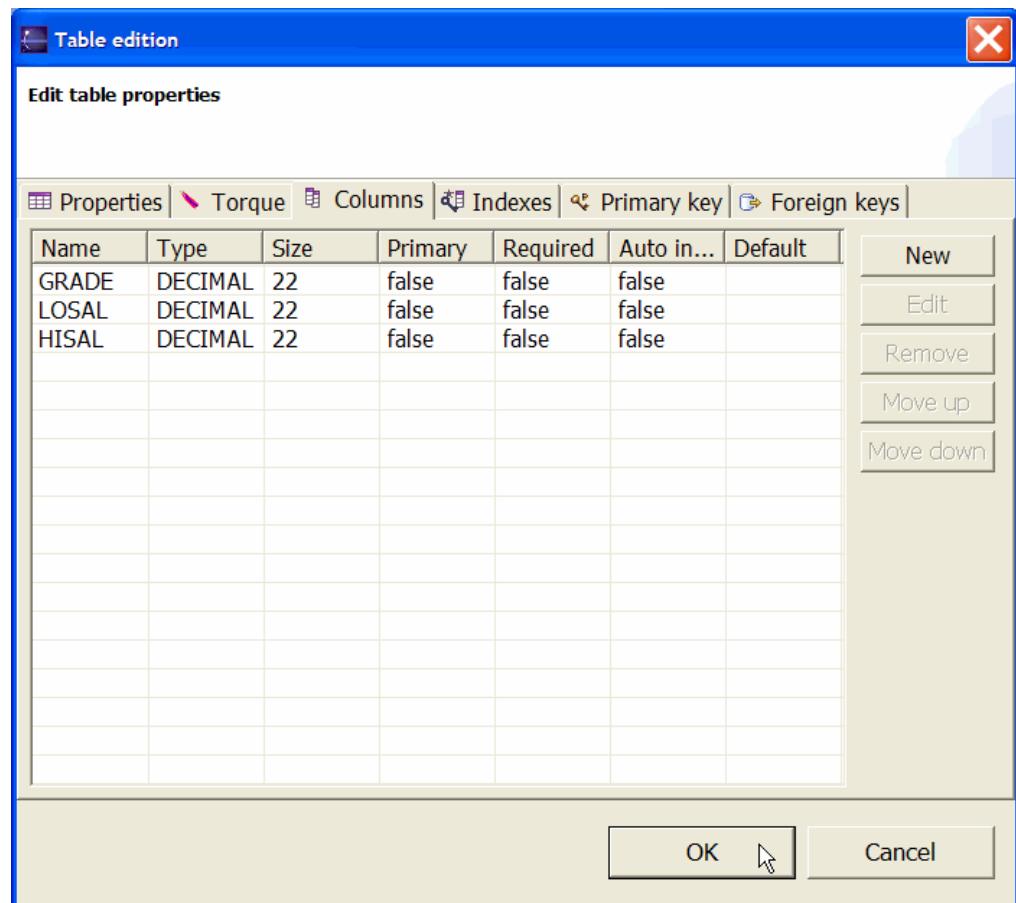


In case you plan to generate Torque objects, you can set the Torque options.

The Database Table Torque page inherits the [Database Diagram Torque Properties](#).

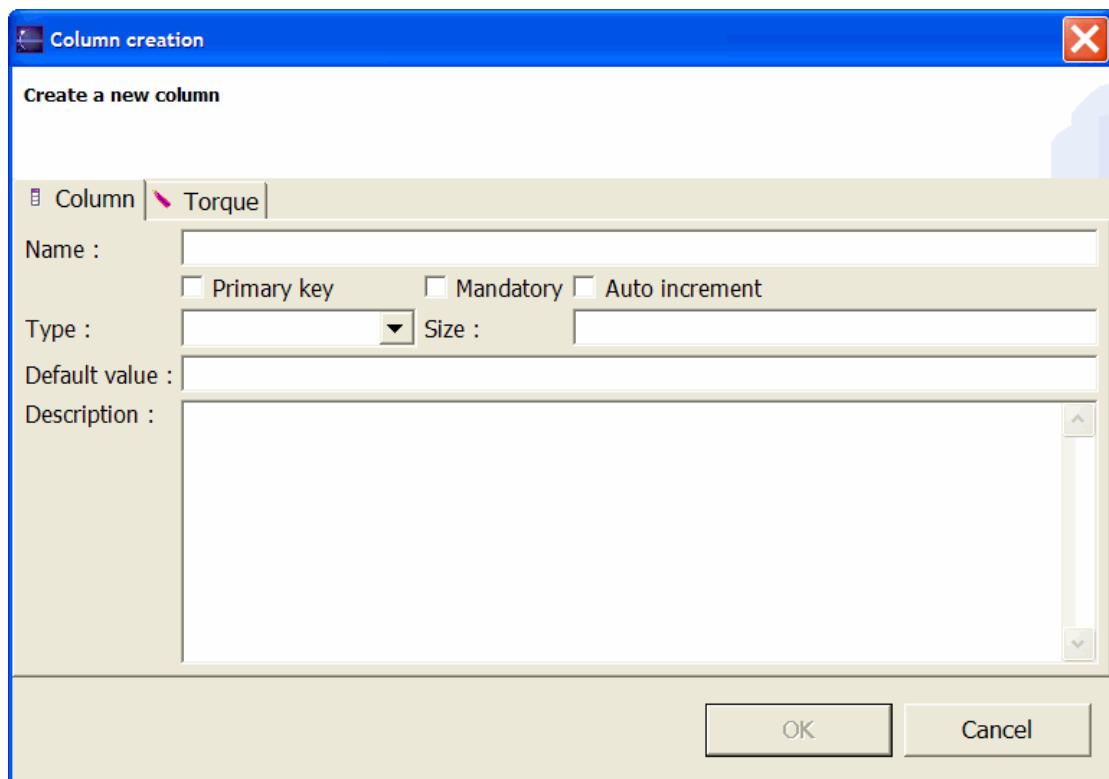
For further details about Torque options, the reader will find a detailed page [here](#).

2.3. Columns



This tabbed folder lets you manage the Database Columns of your current Database Table.

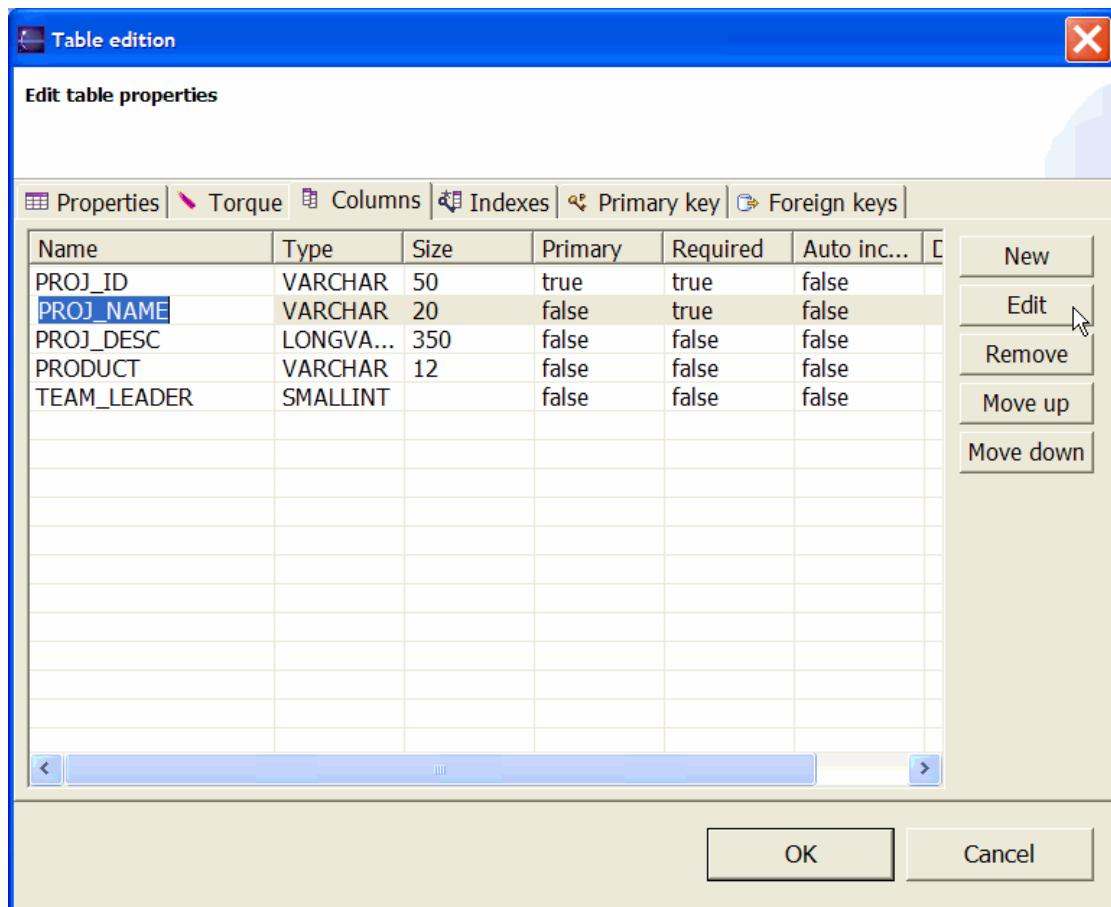
2.3.1. New



This page lets you define a new Database Column in your current Database Table.

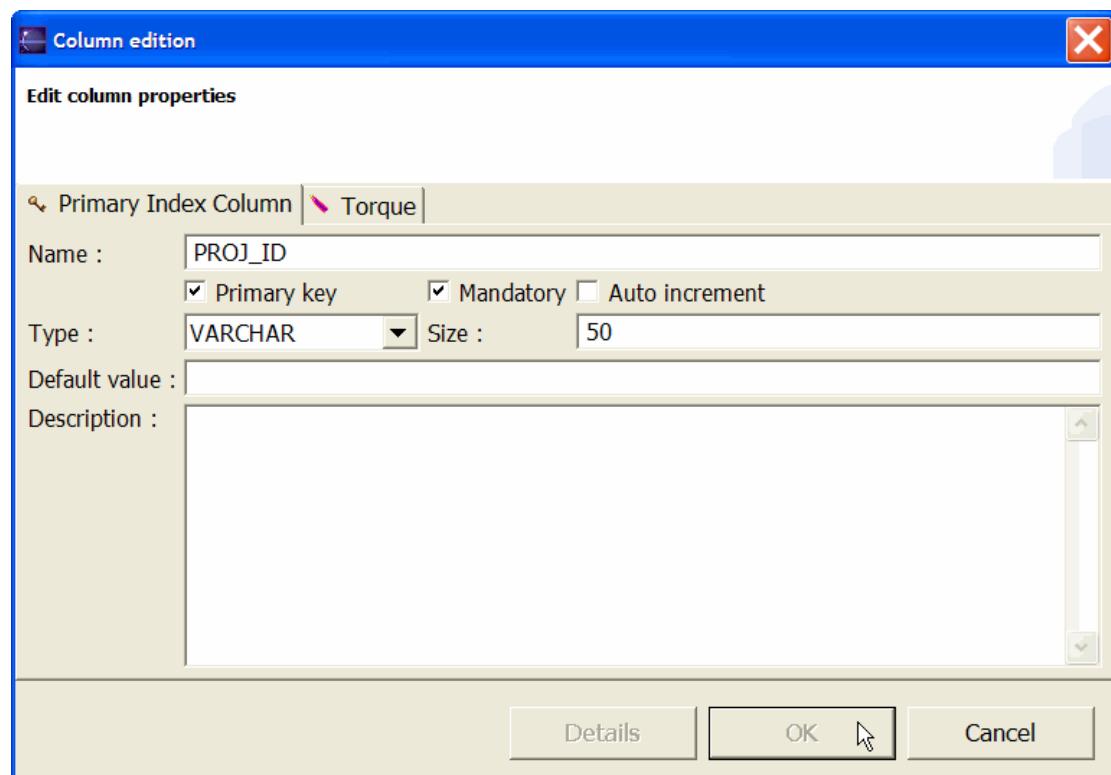
The [Database Column Editor Properties](#) has a complete description of these two tabbed folders.

2.3.2. Edit



Select a Database Column.

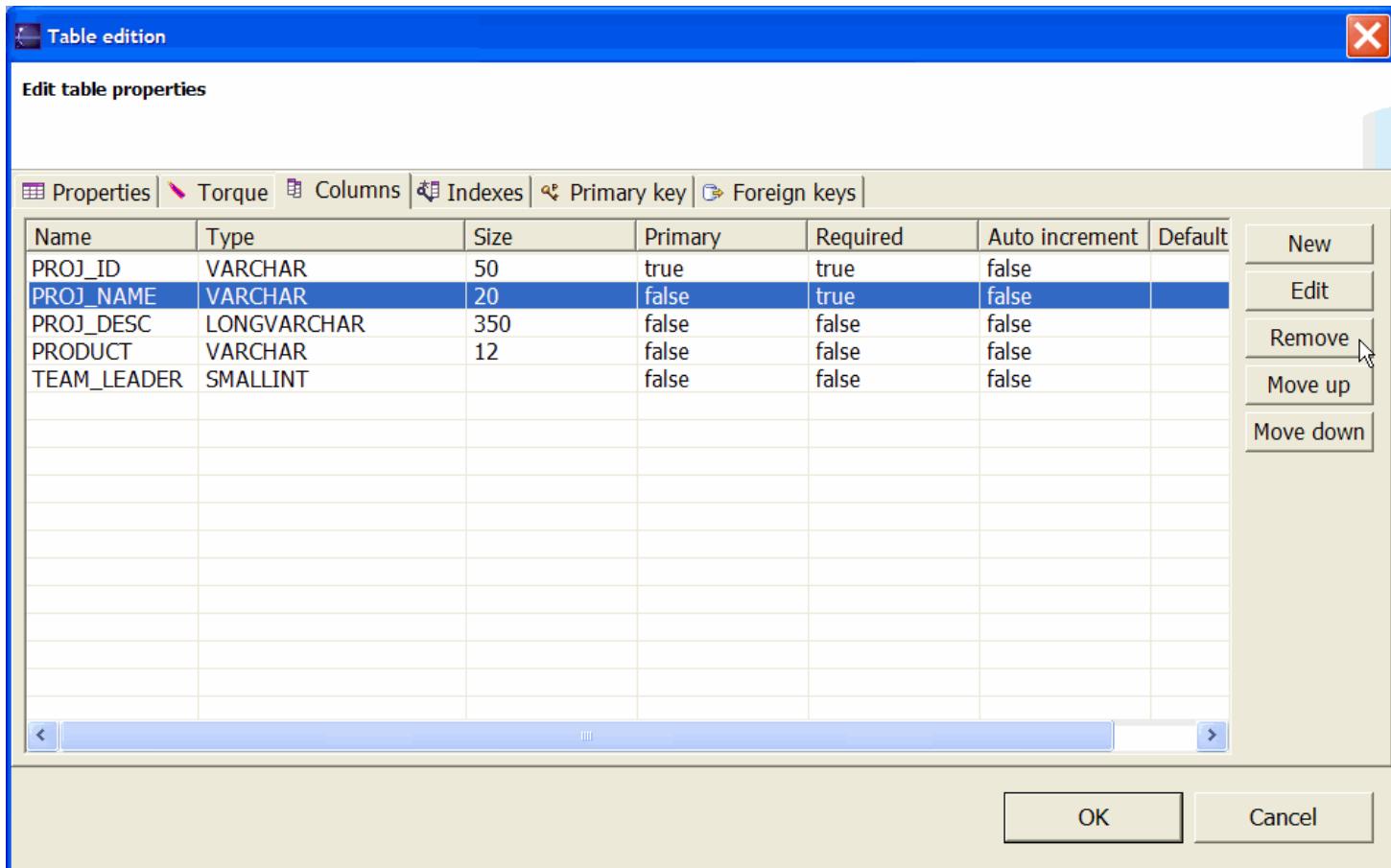
Then use the **Edit** button.



This page lets you update a Database Column in your current Database Table.

The [Database Column Editor Properties](#) has a complete description of these two tabbed folders.

2.3.3. Remove



Select a Database Column.

Then use the **Remove** button.

The selected Database Column will be removed.

2.3.4. Move Up

You can change the Database Column sequence inside your Database Table definition.

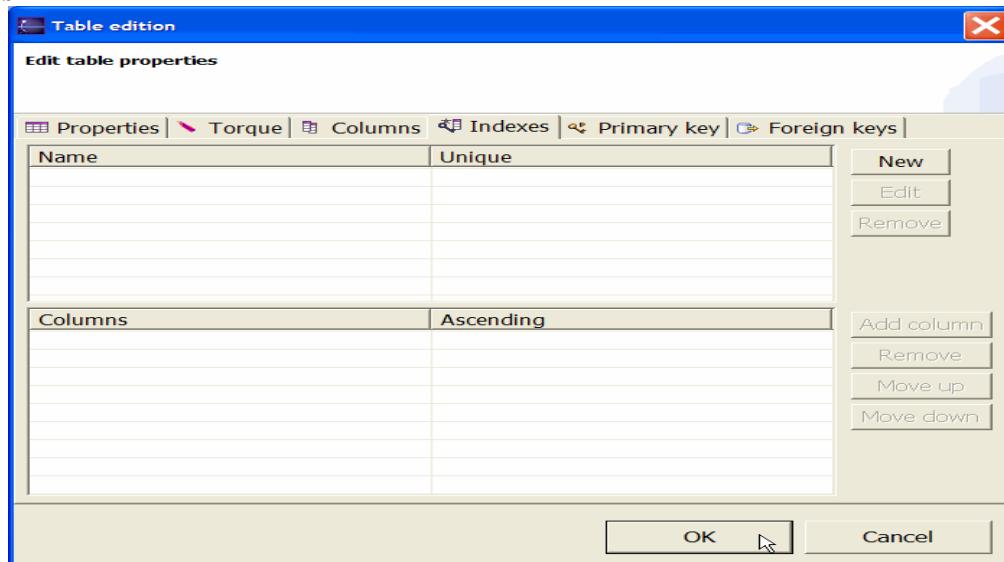
The **Move Up** button moves up the selected Database Column.

2.3.5. Move Down

You can change the Database Column sequence inside your Database Table definition.

The **Move Down** button moves down the selected Database Column.

2.4. Indexes

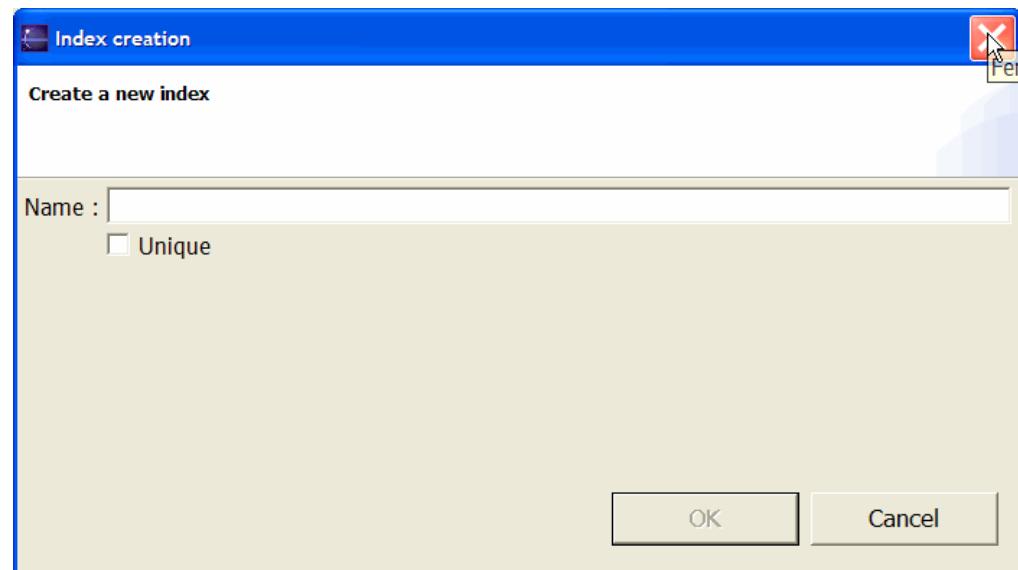


This page has two arrays.

The Upper array contains the Database Index name of the current Database Table.
Once selected, its Database Columns definition will be displayed in the Columns array.

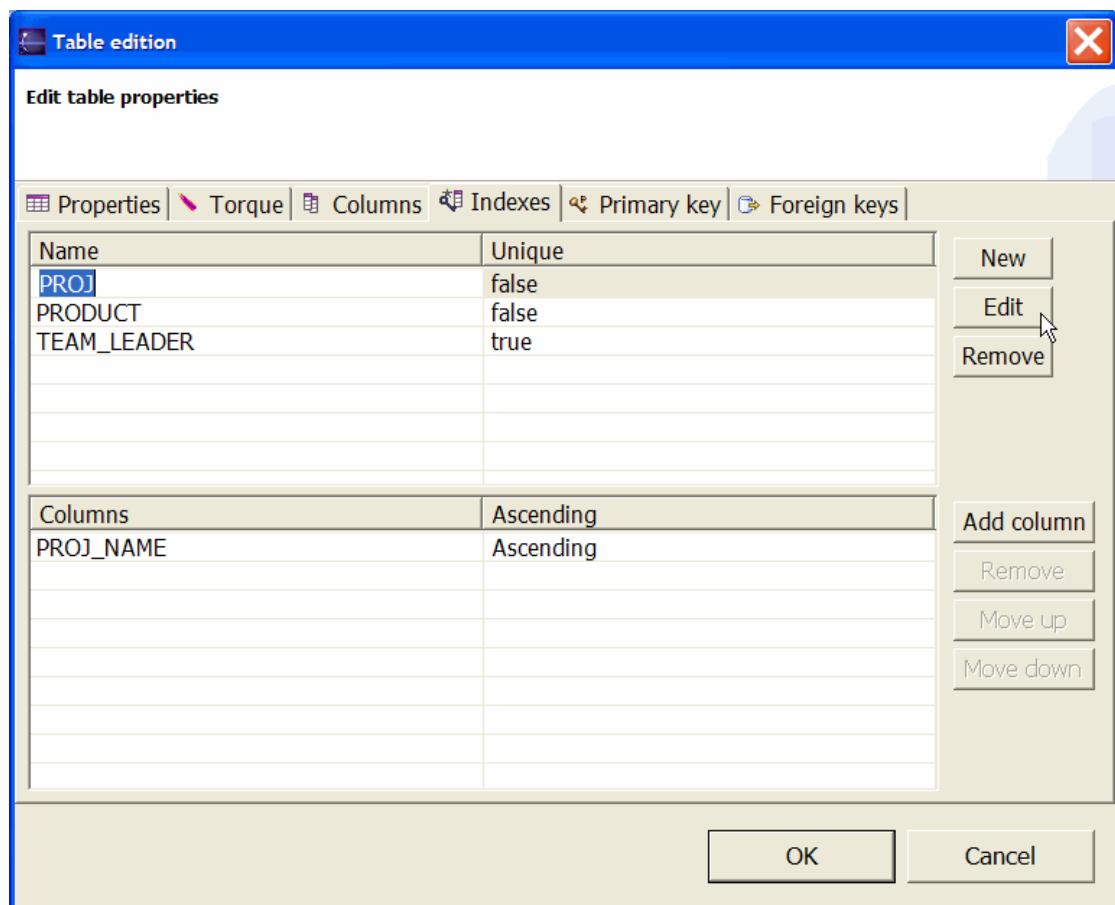
2.4.1. Indexes

2.4.1.1. New

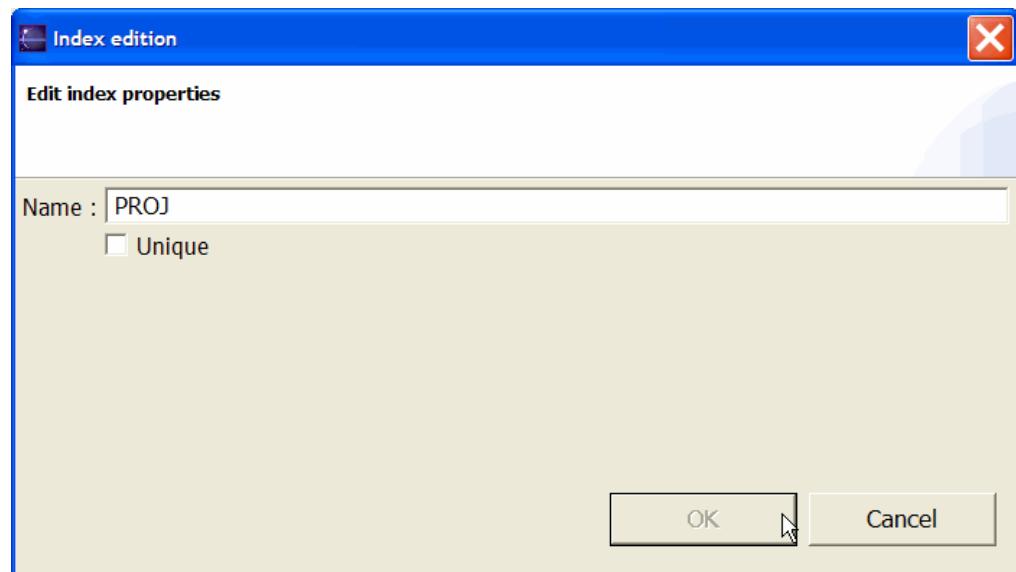


This page lets you define a new Database Index in your current Database Table.
This Database Index could be unique if the **Unique** check box is selected.

2.4.1.2. Edit

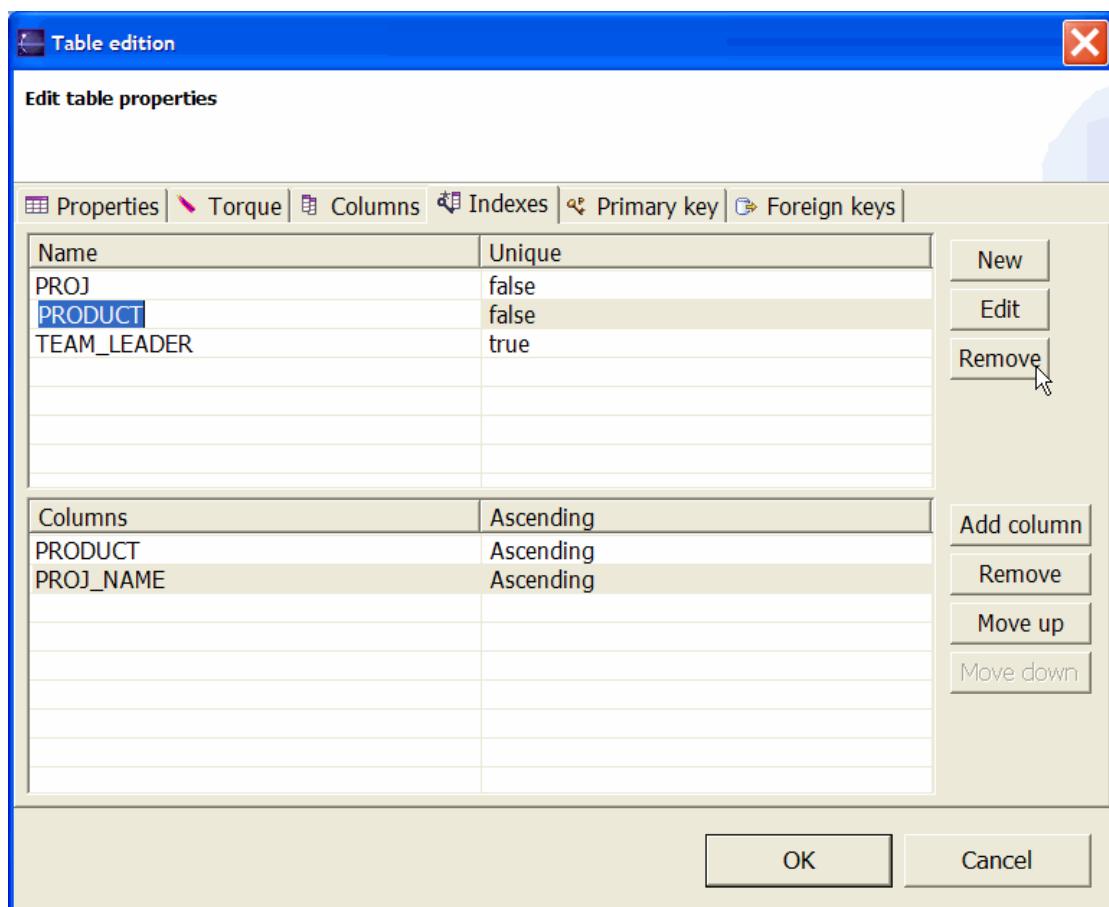


Select a Database Index.
Then use the **Edit** button.



This page lets you update a Database Index in your current Database Table.

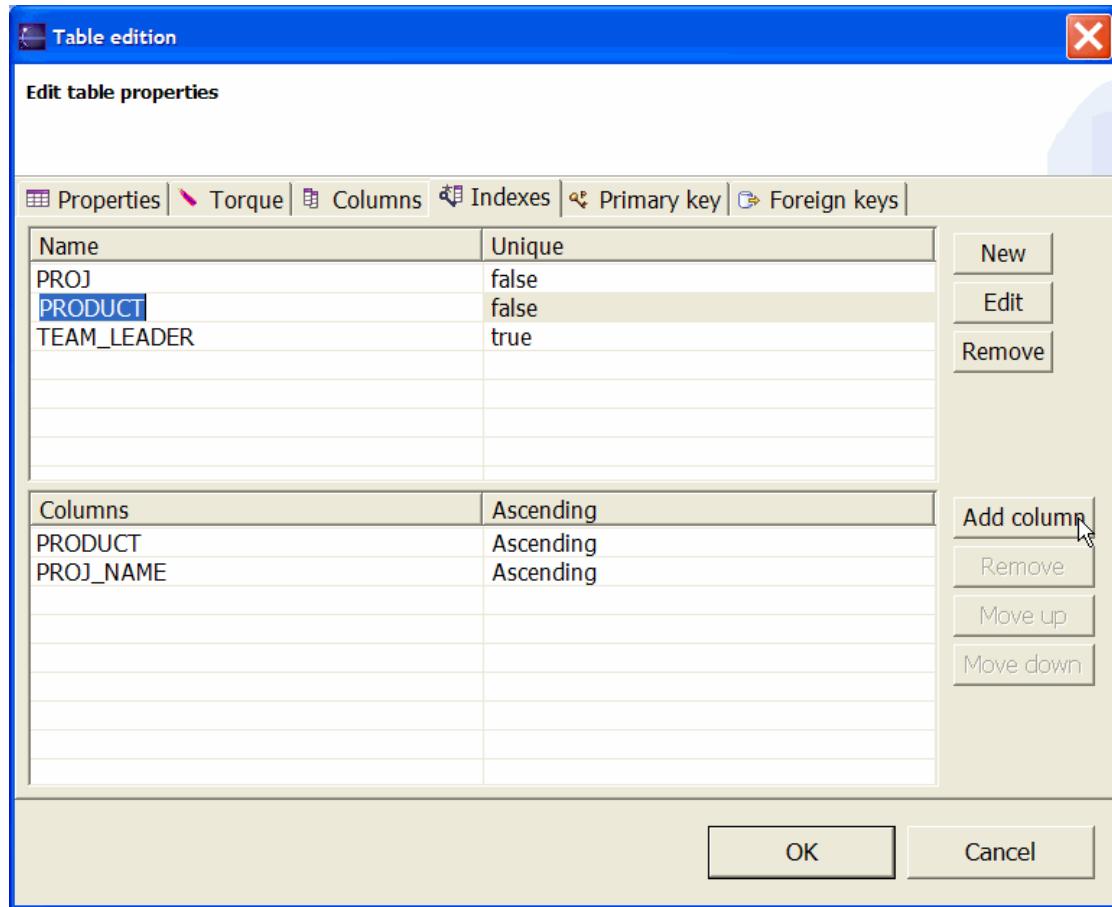
2.4.1.3. Remove



Select a Database Index.
Then use the **Remove** button.
The selected Database Index will be removed.

2.4.2. Columns

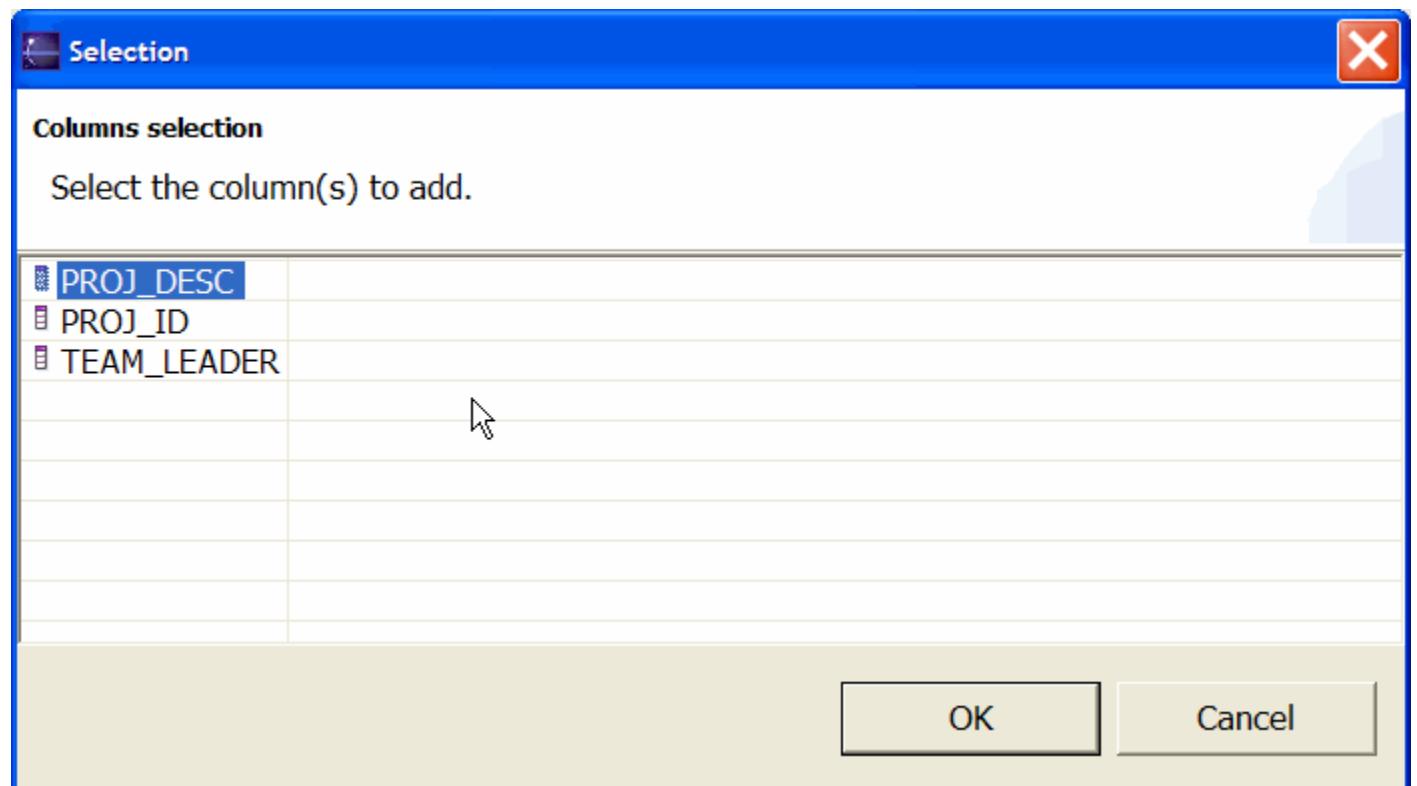
2.4.2.1. Add Column



Select a Database Index.

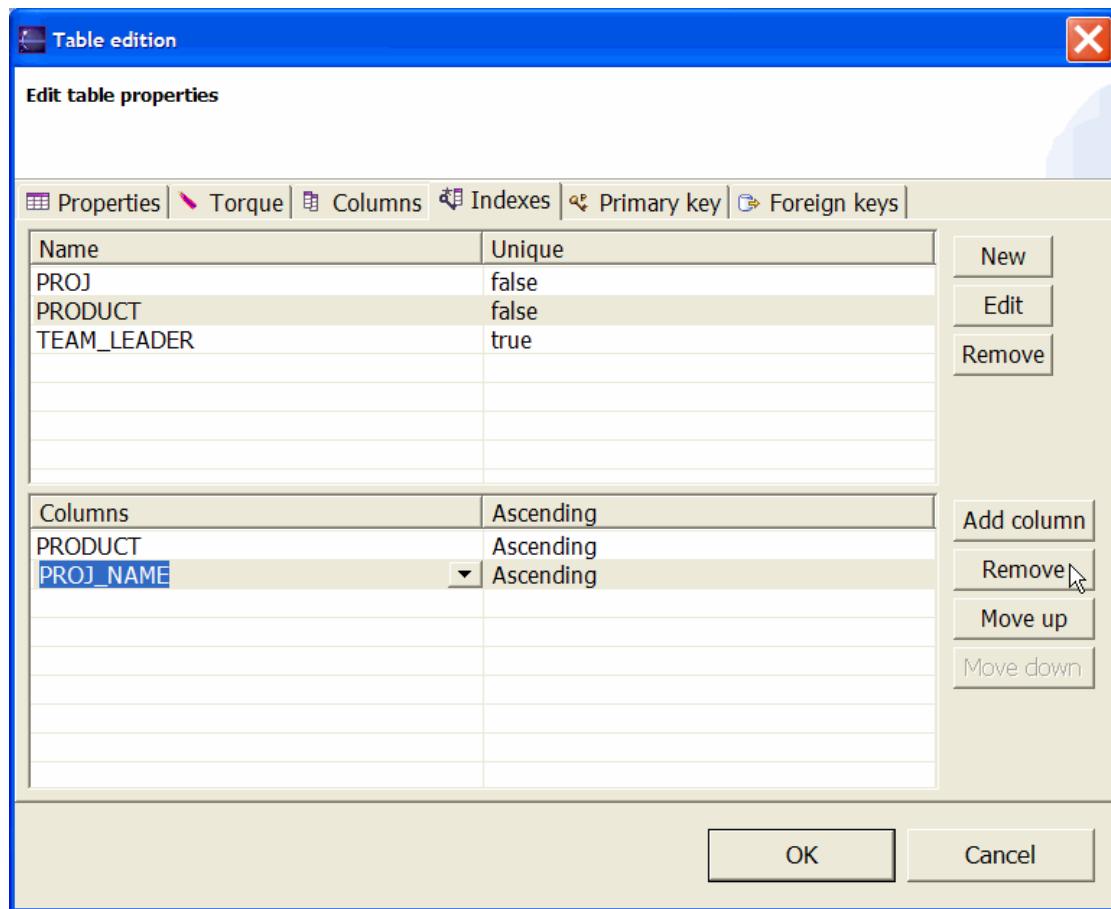
Its Database Columns will be displayed in the Columns array.

Then use the **Add Column** button.



Select the needed Database Column to be part of your current Database Index.

2.4.2.2. Remove



Select a Database Index.

Its Database Columns will be displayed in the Columns array.

Select a Database Column.

Then use the **Remove** button.

The selected Database Column will be removed from your current Database Index.

2.4.2.3. Move Up

You can change the Database Column sequence inside your Database Index definition.

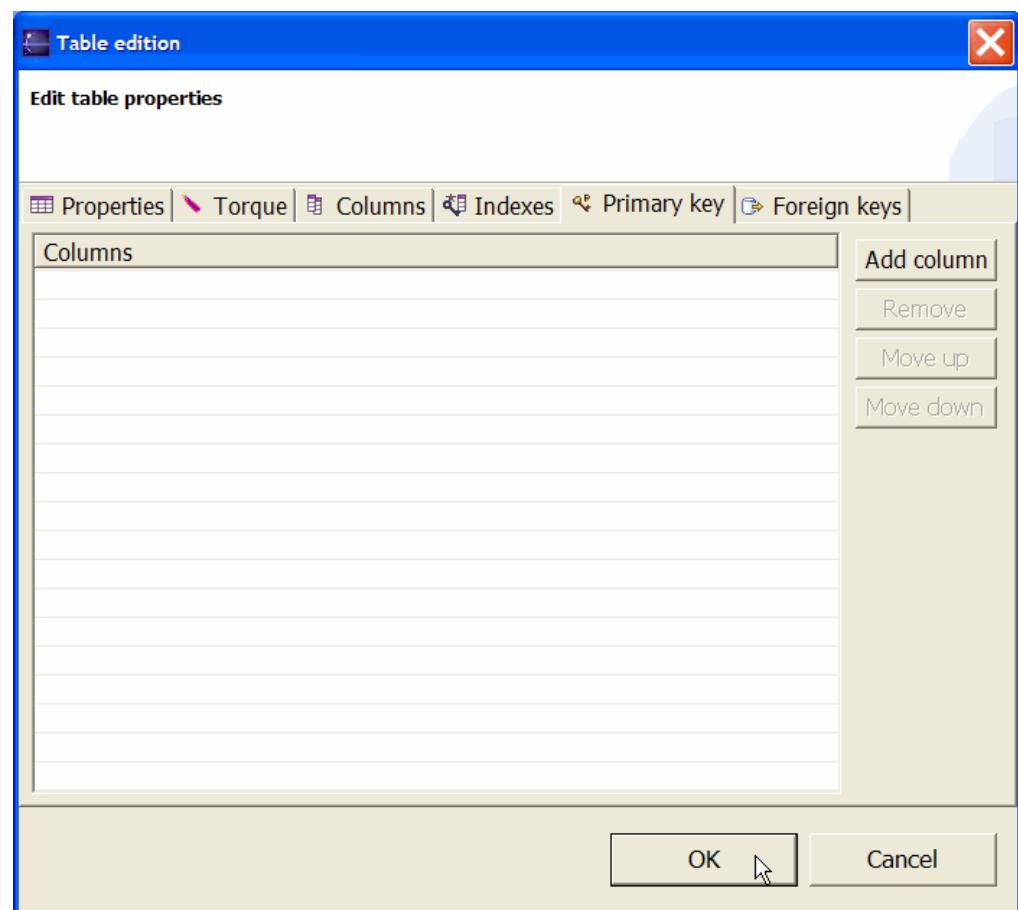
The **Move Up** button moves up the selected Database Column.

2.4.2.4. Move Down

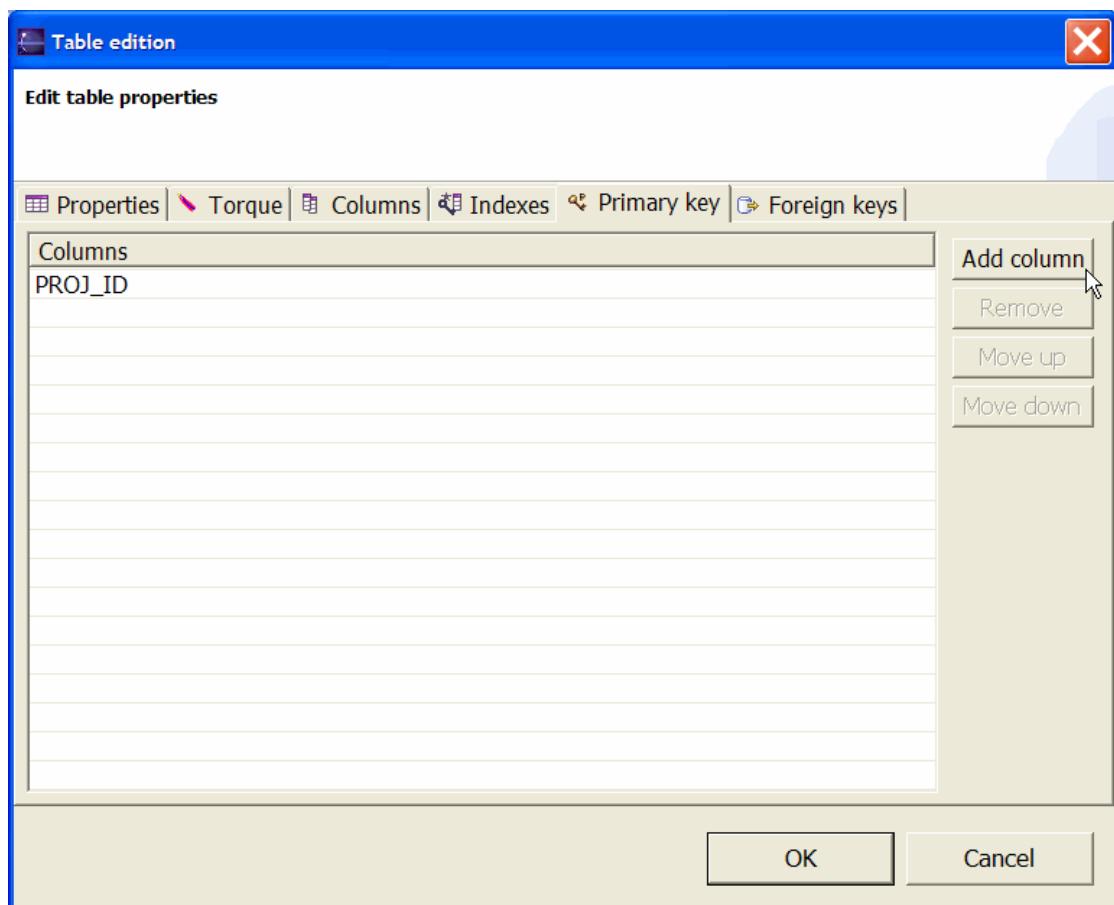
You can change the Database Column sequence inside your Database Index definition.

The **Move Down** button moves down the selected Database Column.

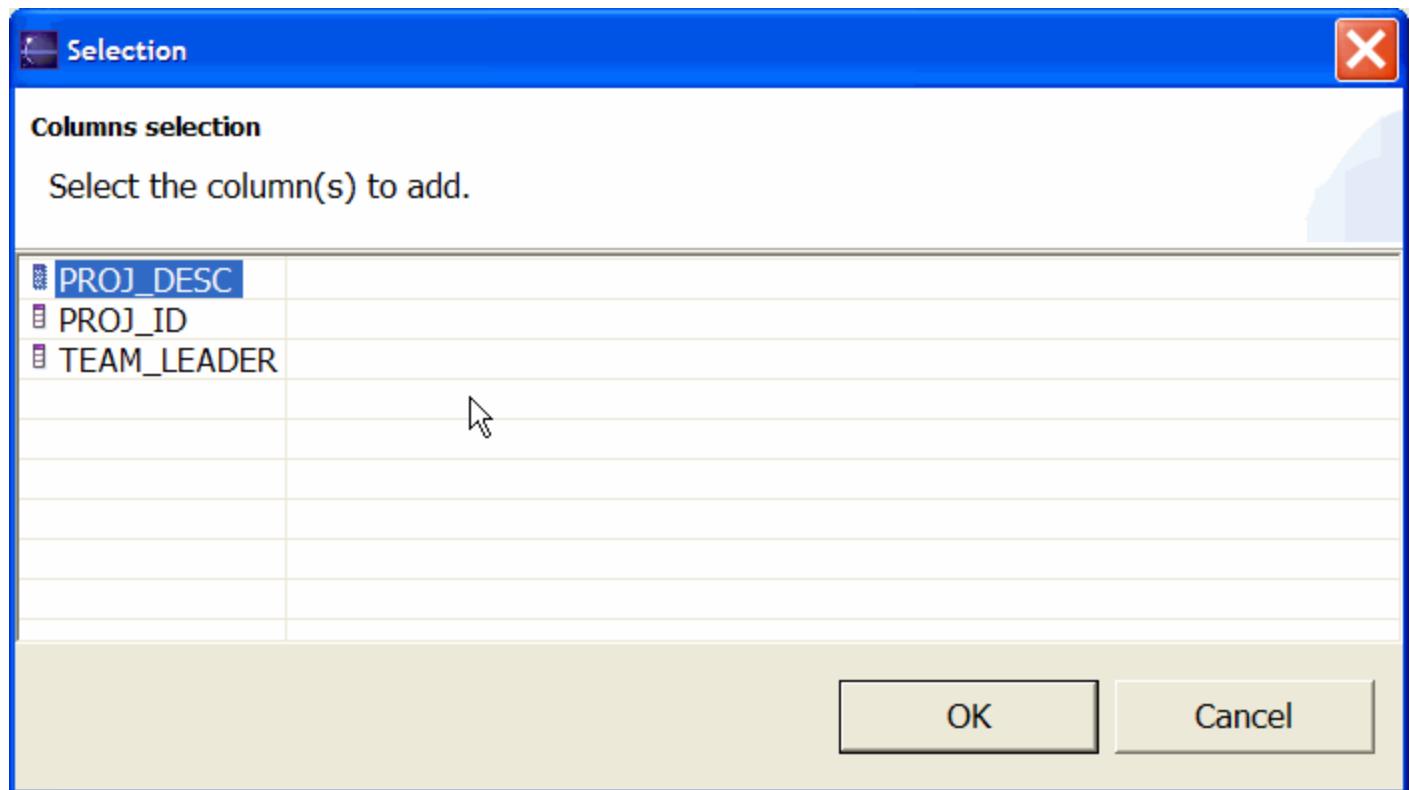
2.5. Primary key



2.5.1. Add Column

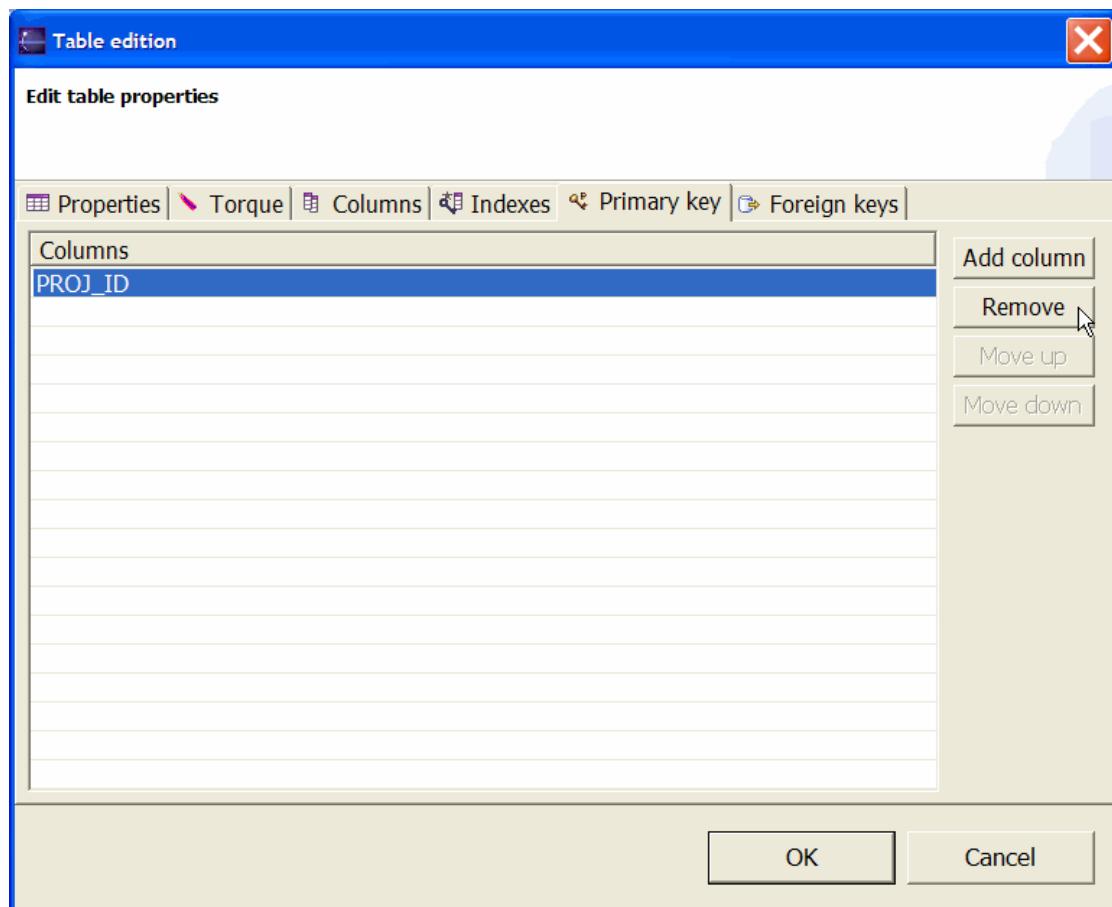


Use the **Add Column** button.



Select the needed Database Column to be part of your current Database Primary Key.

2.5.2. Remove



Select a Database Column.

Then use the **Remove** button.

The selected Database Column will be removed from the current Database Primary Key.

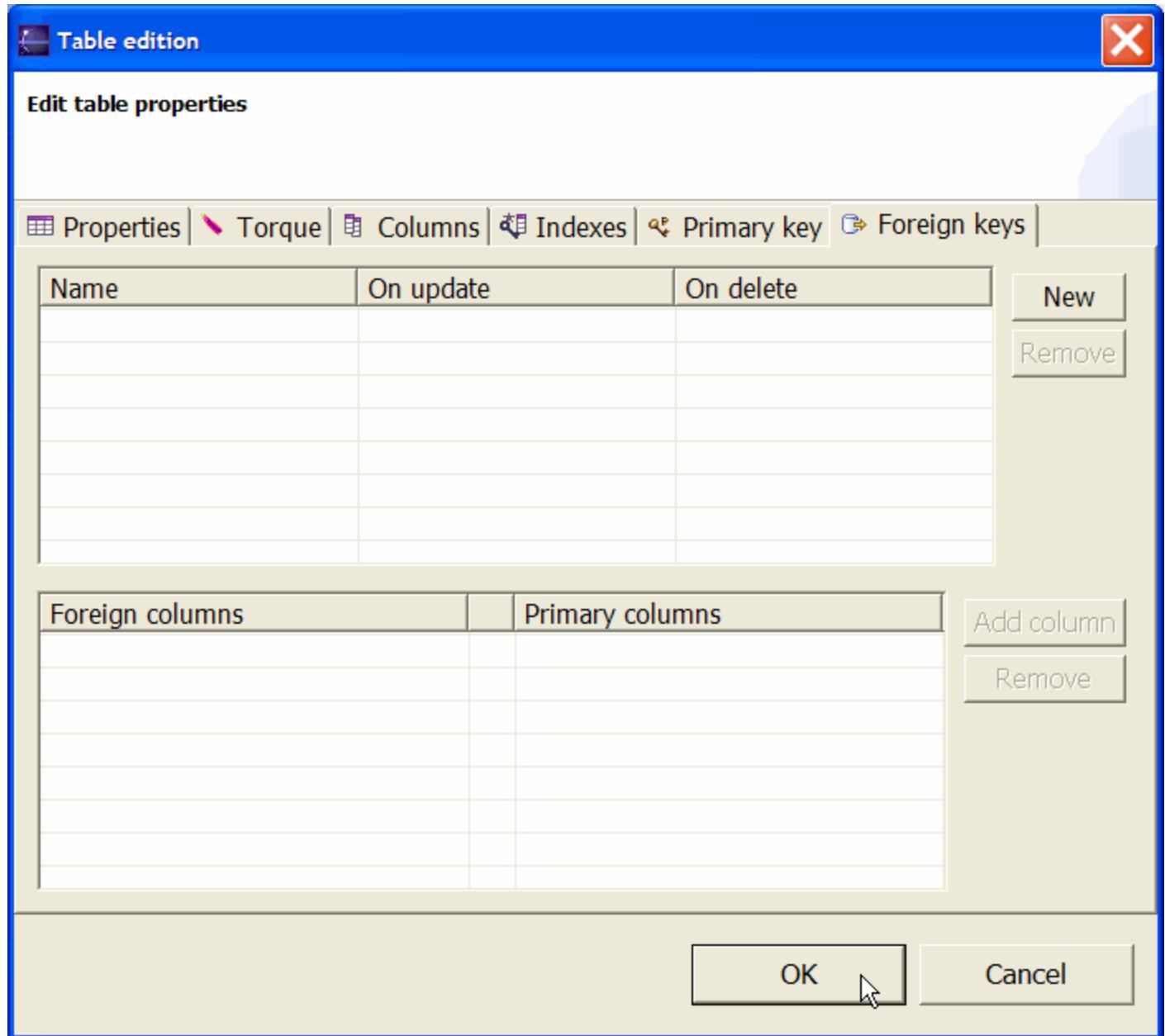
2.5.3. Move Up

You can change the Database Column sequence inside your Database Primary Key definition. The **Move Up** button moves up the selected Database Column.

2.5.4. Move Down

You can change the Database Column sequence inside your Database Primary Key definition. The **Move Down** button moves down the selected Database Column.

2.6. Foreign keys



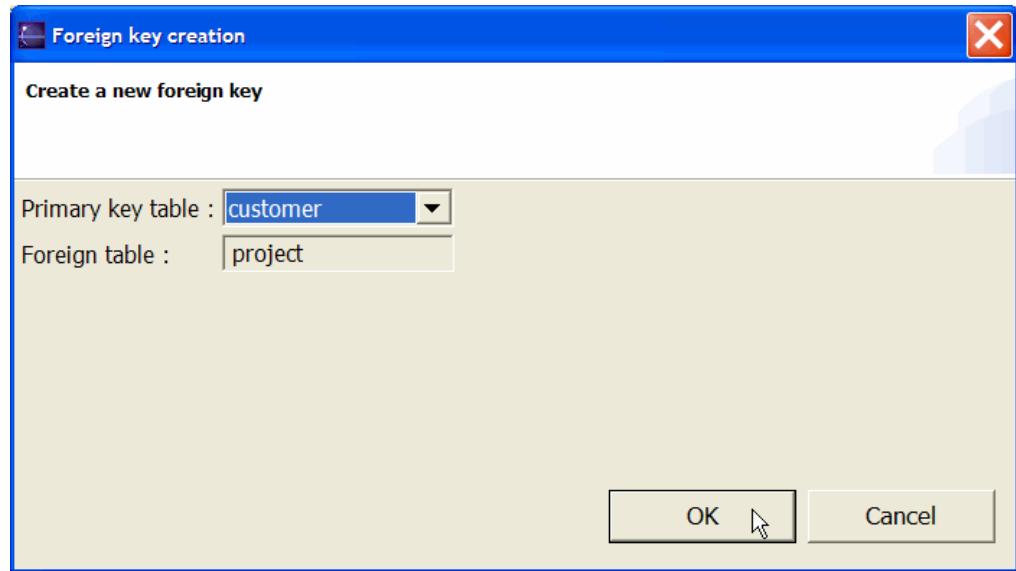
This page has two arrays.

The Upper array contains the Database Foreign Key name of the current Database Table.

Once selected, its Database Foreign Key Columns definition will be displayed in the Foreign Key Columns array.

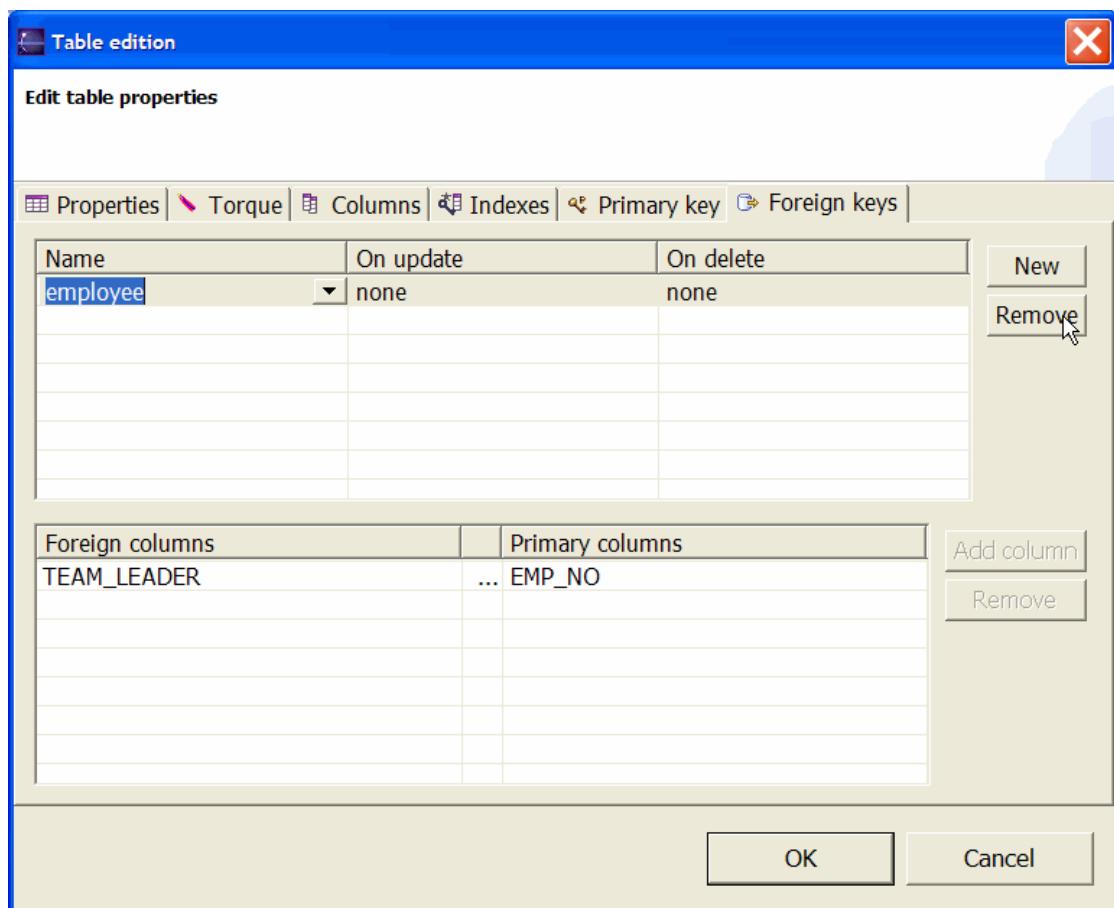
2.6.1. Foreign Keys

2.6.1.1. New



This page lets you define a new Database Foreign Key in your current Database Table. The **Primary Key Table** selector lets you choose a Foreign Primary Key. The **Foreign Table** field is a reminder of current Database Table.

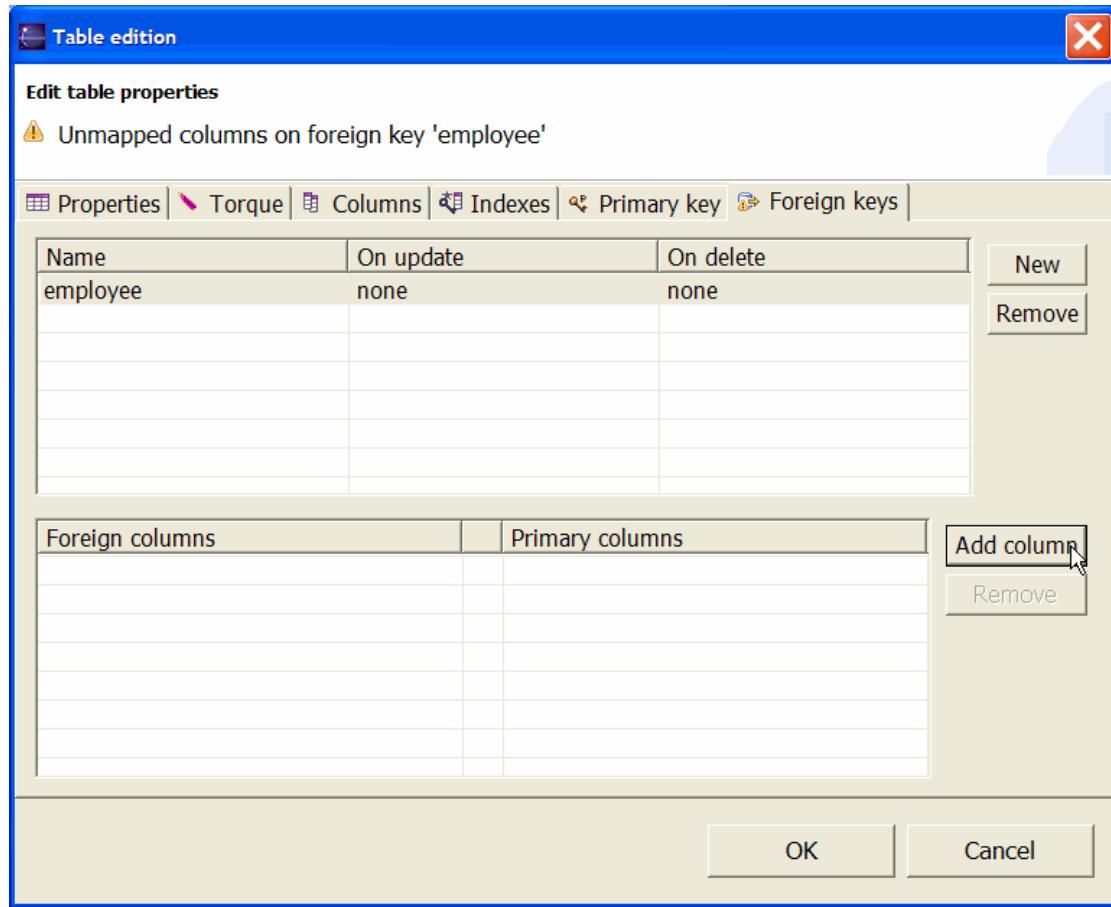
2.6.1.2. Remove



Select a Database Foreign Key.
Then use the **Remove** button.
The selected Database Foreign Key will be removed.

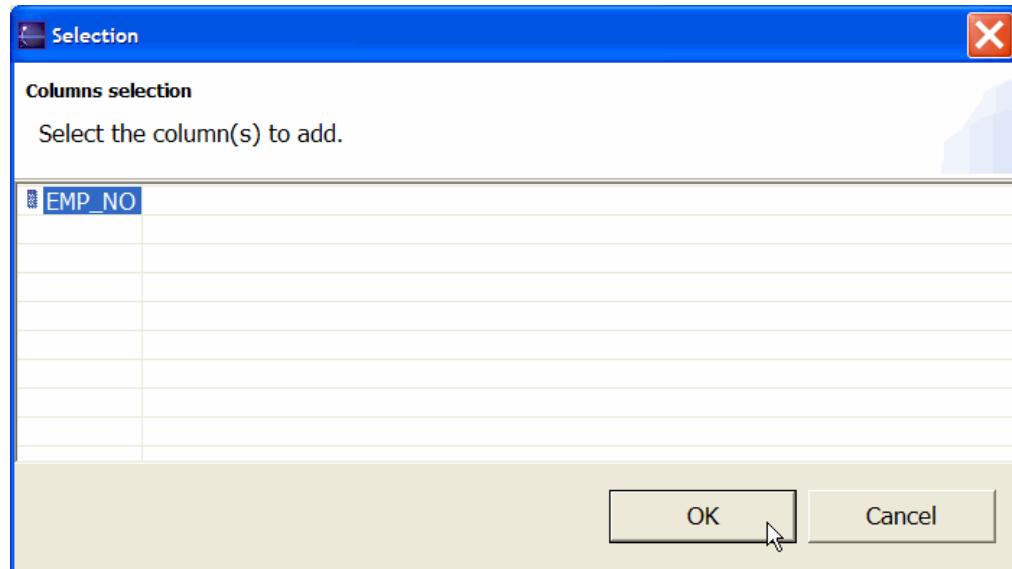
2.6.2. Foreign Columns

2.6.2.1. Add Column

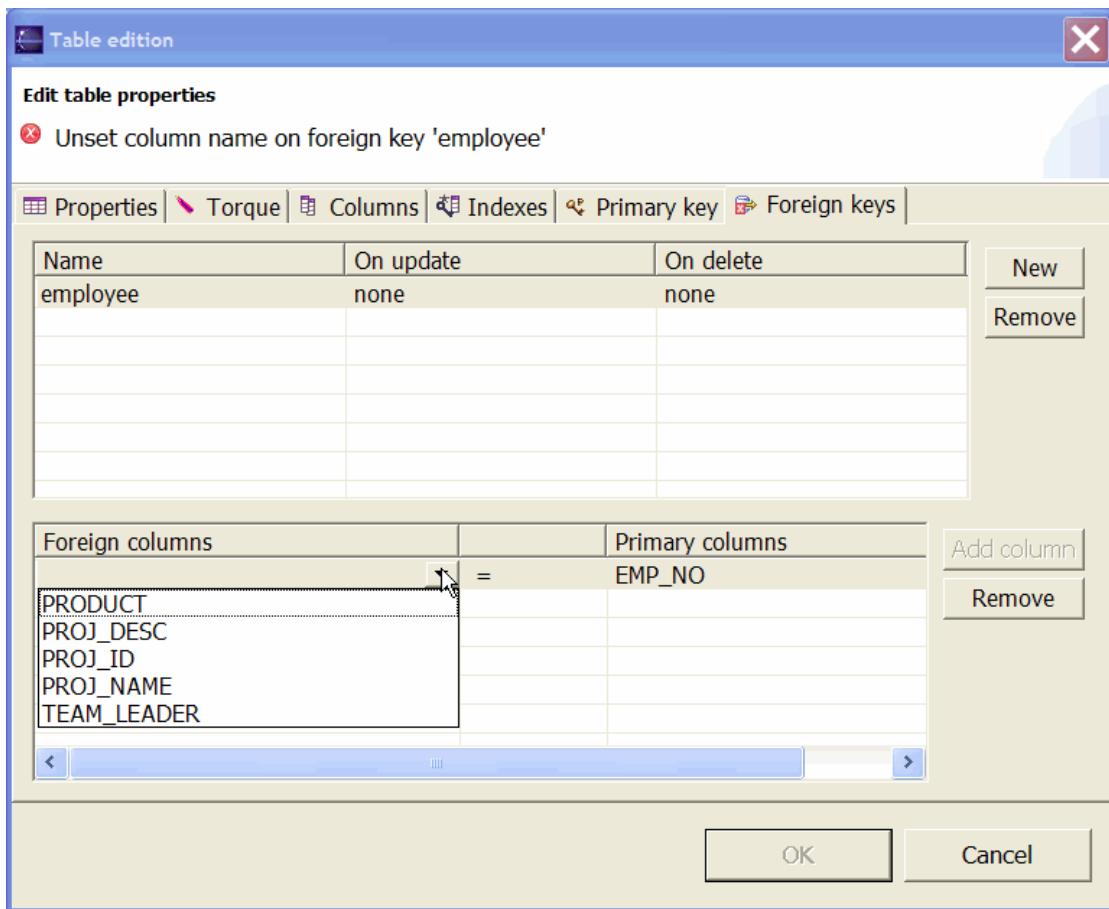


Select a Database Foreign Key.

Its Database Foreign Key Columns definition will be displayed in the Foreign Key Columns array. Then use the **Add Column** button.

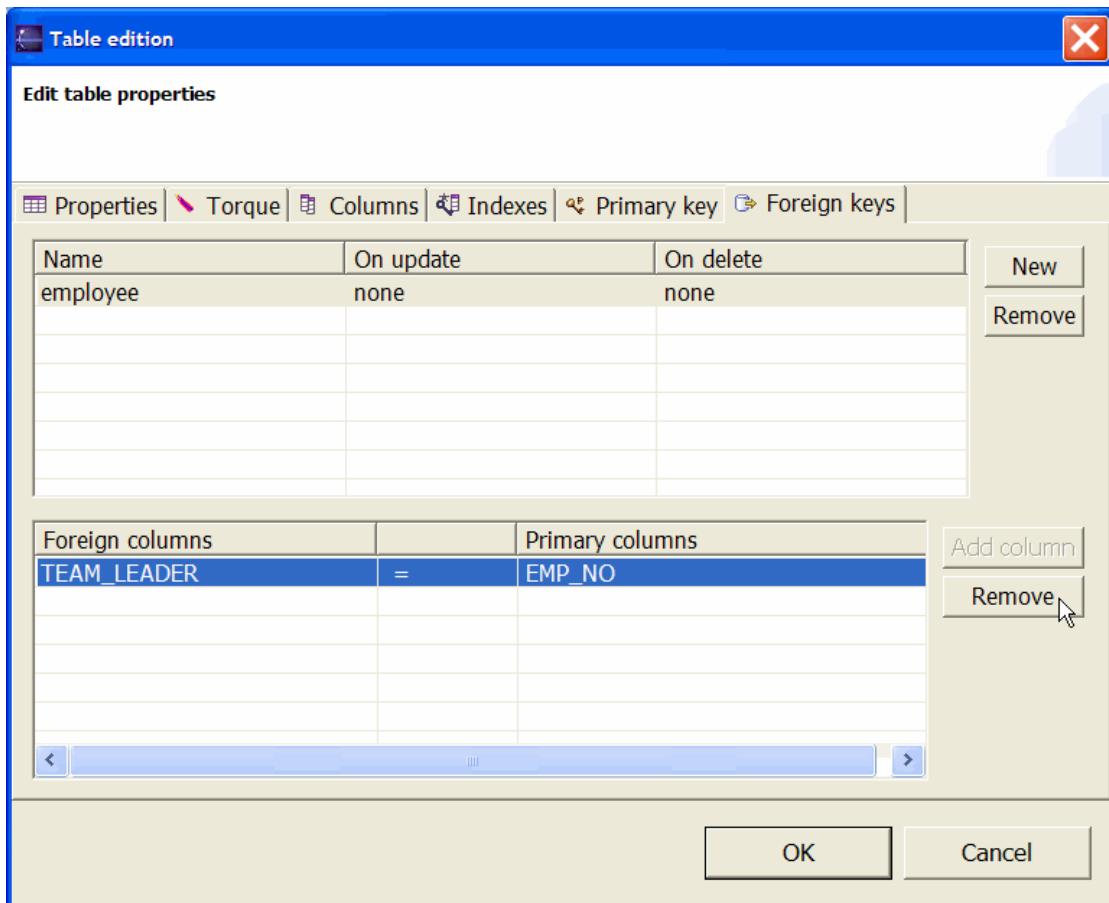


Select the needed Foreign Database Primary Key Column to be part of your current Database Foreign Key definition.



The Foreign Primary Key Column is added to your Foreign Key Columns definition array. Then select the local Database Column mapped to the Foreign Primary Key Column.

2.4.2.2. Remove



Select a Database Foreign Key..

Its Database Foreign Key Columns will be displayed in the Foreign Key Columns definition array.

Select a Foreign Key Column.

Then use the **Remove** button.

The selected Database Foreign Key Column will be removed from the current Database Foreign Key.

2.7. OK

Once you have decided to update your Database Column, click the **OK** button.

The update will be done.

A detailed output will be displayed in the [DatabaseConsole](#).

```
Properties Database Console x
2004-10-05 15:43:55 Schema schema->Table project->Foreign Key employee->Table project
2004-10-05 15:43:55 Schema schema->Table employee->Foreign Key Reference project->Table project
2004-10-05 15:43:55 Schema schema->Table project->Foreign Key Reference employee_project->Table project
2004-10-05 15:43:55 Schema schema->Table employee_project->Foreign Key project->Table project
2004-10-05 15:43:55 Schema schema->Table project->Foreign Key Reference proj_dept_budget->Table project
2004-10-05 15:43:55 Schema schema->Table proj_dept_budget->Foreign Key project->Table project
2004-10-05 15:43:55 Schema schema->Table project : Updated
2004-10-05 15:43:55 Table project->Primary Key->Column PROJ_ID
2004-10-05 15:43:55 Table project->Foreign Key Reference employee_project->Table project->Primary Key->Column PROJ_ID
2004-10-05 15:43:55 Table project->Foreign Key Reference employee_project->Table employee_project->Column PROJ_ID [ ]
2004-10-05 15:43:55 Table employee_project->Foreign Key project->Table project->Primary Key->Column PROJ_ID
2004-10-05 15:43:55 Table employee_project->Foreign Key project->Table employee_project->Column PROJ_ID
2004-10-05 15:43:55 Table employee_project->Column PROJ_ID
2004-10-05 15:43:55 Table employee_project->Primary Key->Column PROJ_ID
2004-10-05 15:43:55 Table employee_project->Unique Index PROJ_ID->Column PROJ_ID
2004-10-05 15:43:55 Table project->Foreign Key Reference proj_dept_budget->Table project->Primary Key->Column PROJ_ID
2004-10-05 15:43:55 Table project->Foreign Key Reference proj_dept_budget->Table proj_dept_budget->Column PROJ_ID
2004-10-05 15:43:55 Table proj_dept_budget->Foreign Key project->Table project->Primary Key->Column PROJ_ID
2004-10-05 15:43:55 Table proj_dept_budget->Foreign Key project->Table proj_dept_budget->Column PROJ_ID
2004-10-05 15:43:55 Table proj_dept_budget->Column PROJ_ID
2004-10-05 15:43:55 Table proj_dept_budget->Primary Key->Column PROJ_ID
2004-10-05 15:43:55 Table proj_dept_budget->Unique Index PROJ_ID->Column PROJ_ID
2004-10-05 15:43:55 Table project->Column PROJ_ID : Updated
2004-10-05 15:43:55 Table project->Non Unique Index PROJ->Column PROJ_NAME
2004-10-05 15:43:55 Table project->Non Unique Index PRODUCT->Column PROJ_NAME
2004-10-05 15:43:55 Table project->Column PROJ_NAME : Updated
2004-10-05 15:43:55 Table project->Column PROJ_DESC : Updated
2004-10-05 15:43:55 Table project->Column PROJ_DESC : Updated
```

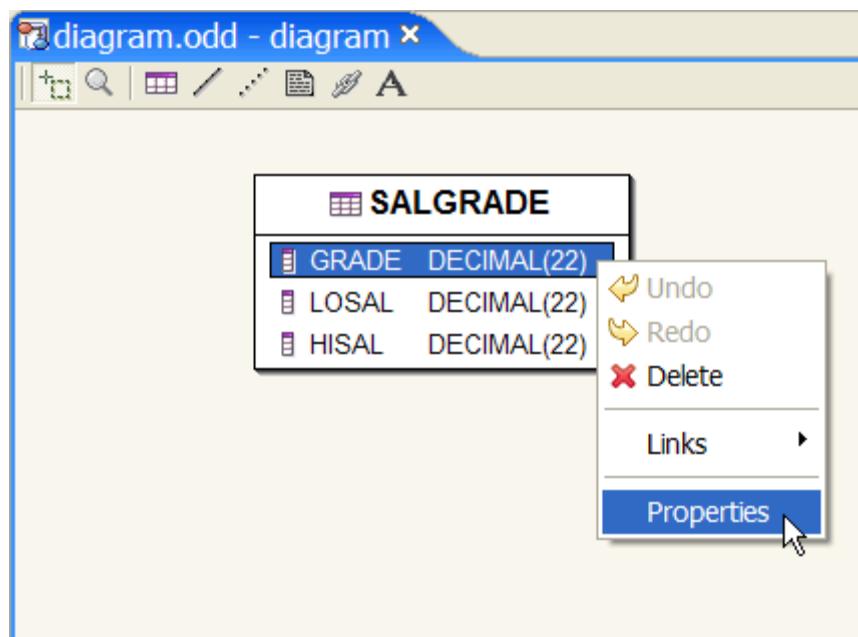
Database Editor Properties - Column

1. Introduction
2. Column
 1. Properties
 1. Name
 2. Primary Key
 3. Mandatory
 4. Auto increment
 5. Type
 6. Size
 7. Default Value
 8. Description
 2. Torque
 3. Details
 4. OK

1. Introduction

In this section you will learn how to use Database Editor Column properties.

2. Column

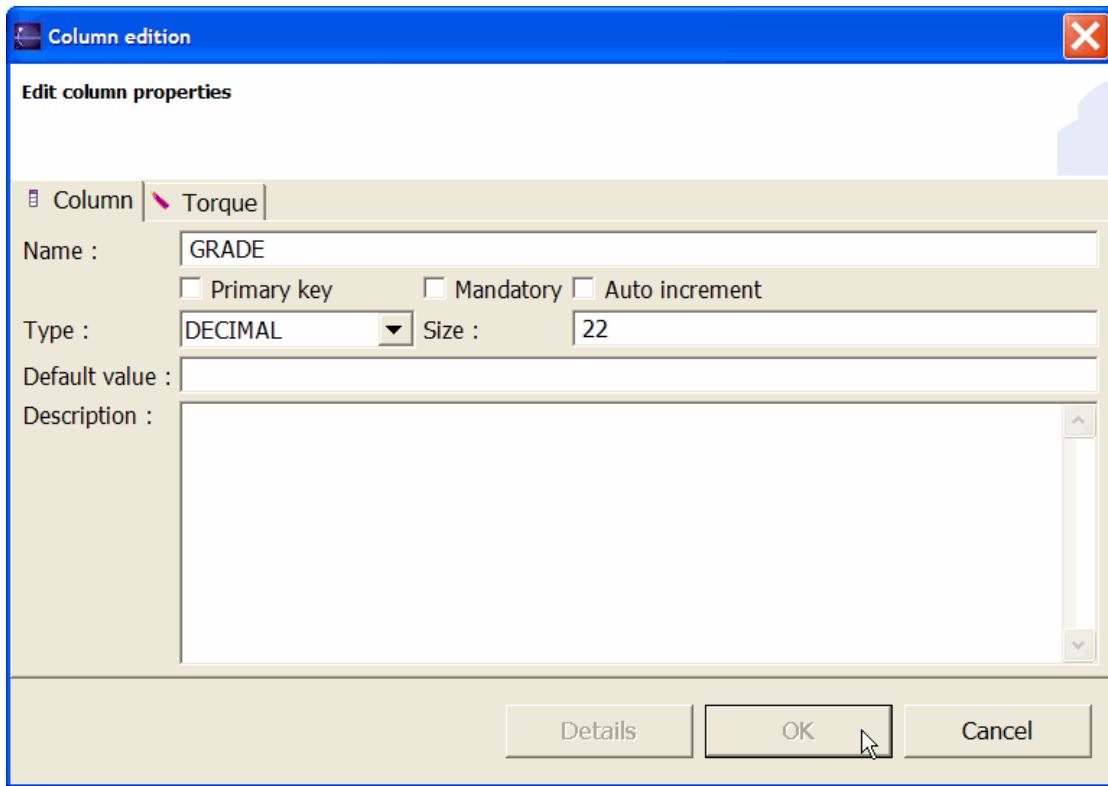


Select a Database Column, right-click and select :

Properties

or

Double Click a Database Column :



Two Tabbed folders are available :

- Properties
- Torque

2.1. Properties

Column properties.

2.1.1. Name

Column name.

2.1.2. Primary key

If set, this Column is part of the current Database Table Primary Key definition.

2.1.3. Mandatory

If set, this Column is mandatory.

2.1.4. Auto increment

If set, this Column is part of a Database Auto Increment process.

2.1.5. Type

Column Type.

2.1.6. Size

Column Size.

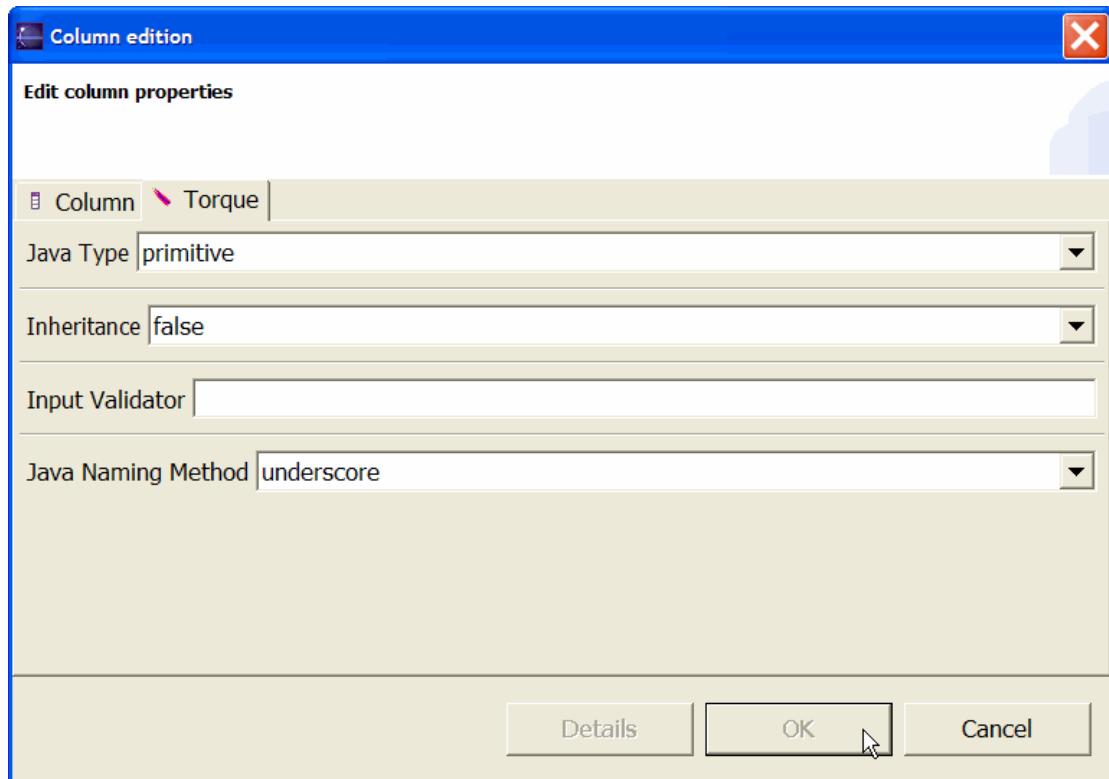
2.1.7. Default Value

Column Default Value.

2.1.8. Description

Column description.

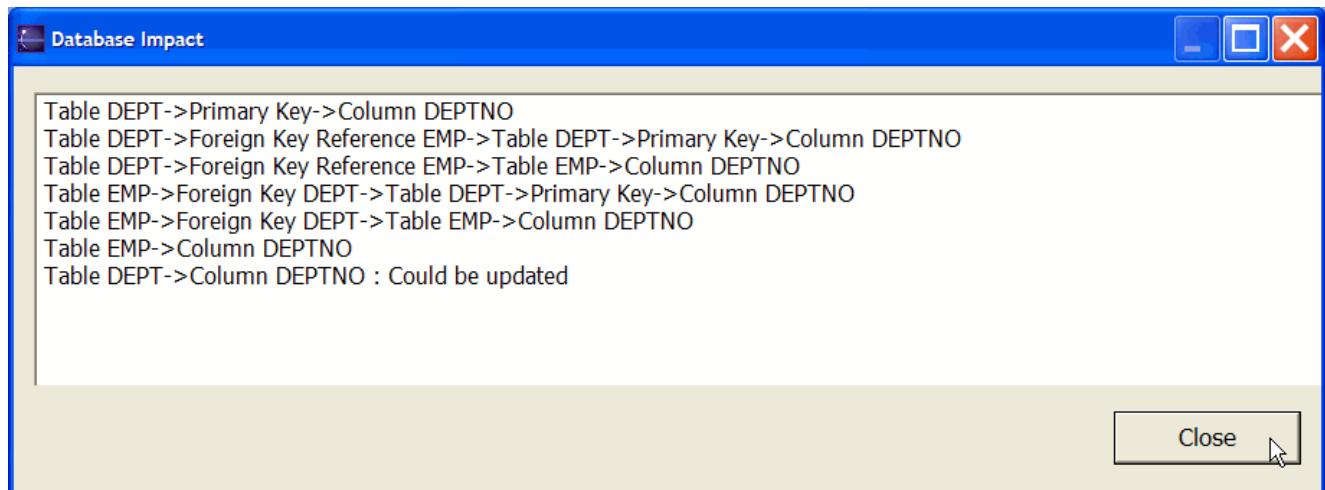
2.2. Torque



In case you plan to generate Torque objects, you can set the Torque options.
For further details about Torque options, the reader will find a detailed page [here](#).

2.3. Details

Once you have modified a Database Column properties, the **Details** button is available.
This button will display a Dialog box who contains the update impact on your Database.

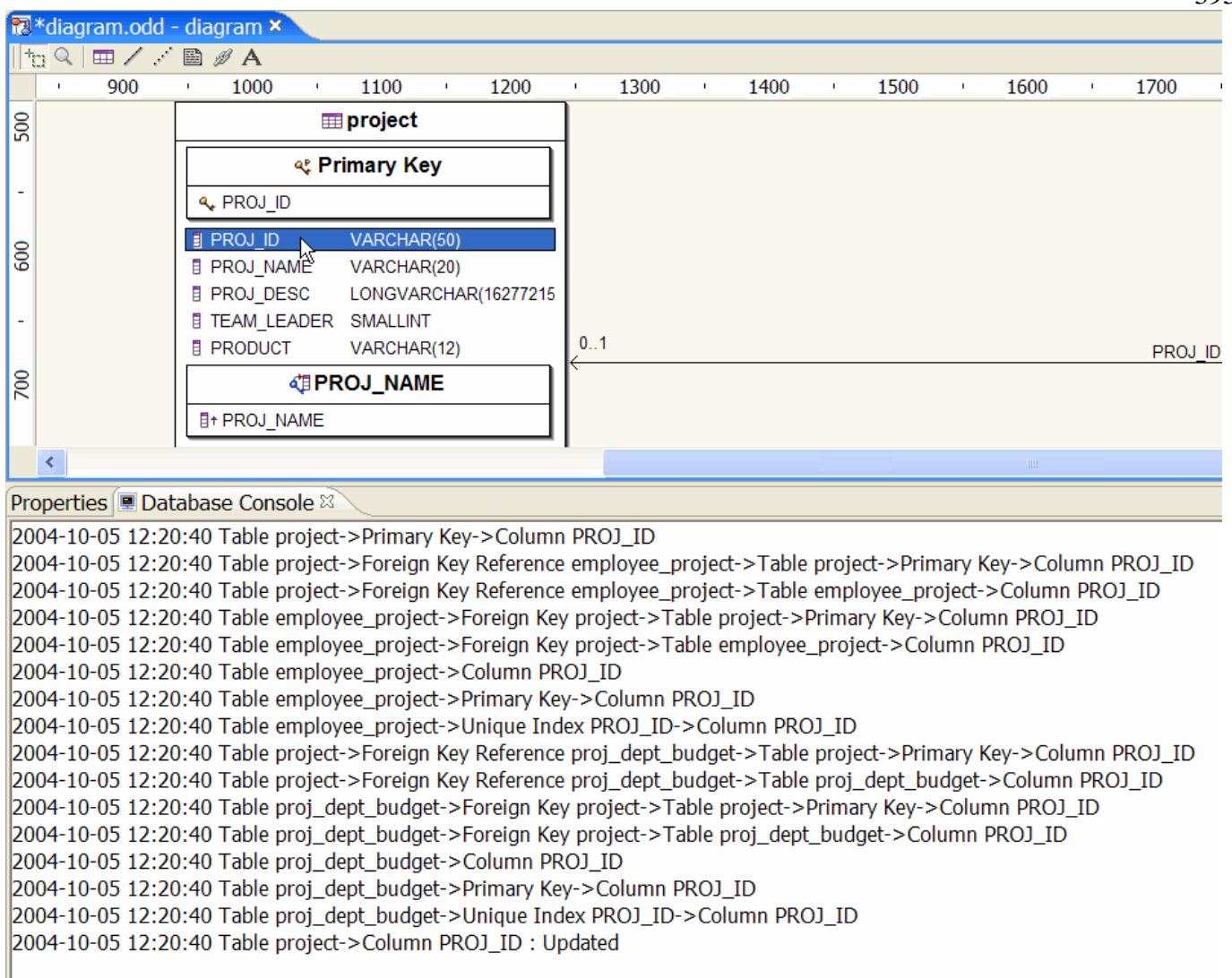


In some cases, the Column Update could fail.
Database Impact will show you a detailed report.

2.4. OK

Once you have decided to update your Database Column, click the **OK** button.
The update will be done.

A detailed output will be displayed in the [DatabaseConsole](#).



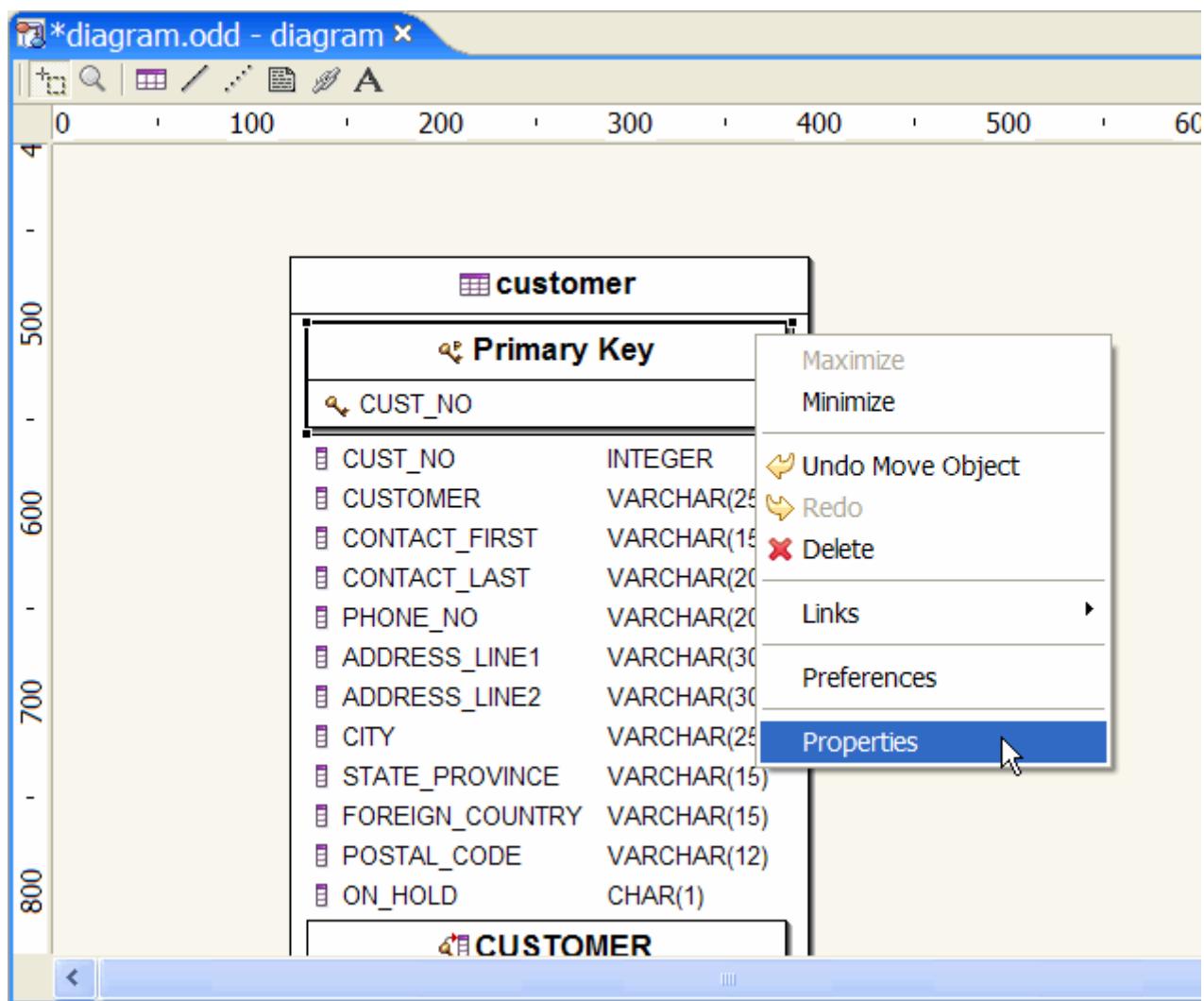
Database Editor Properties – Primary Key

1. Introduction
2. Primary Key
 1. Properties
 1. Add Column
 2. Remove
 3. Move Up
 4. Move Down
 2. OK

1. Introduction

In this section you will learn how to use Database Editor Primary Key properties.

2. Primary Key

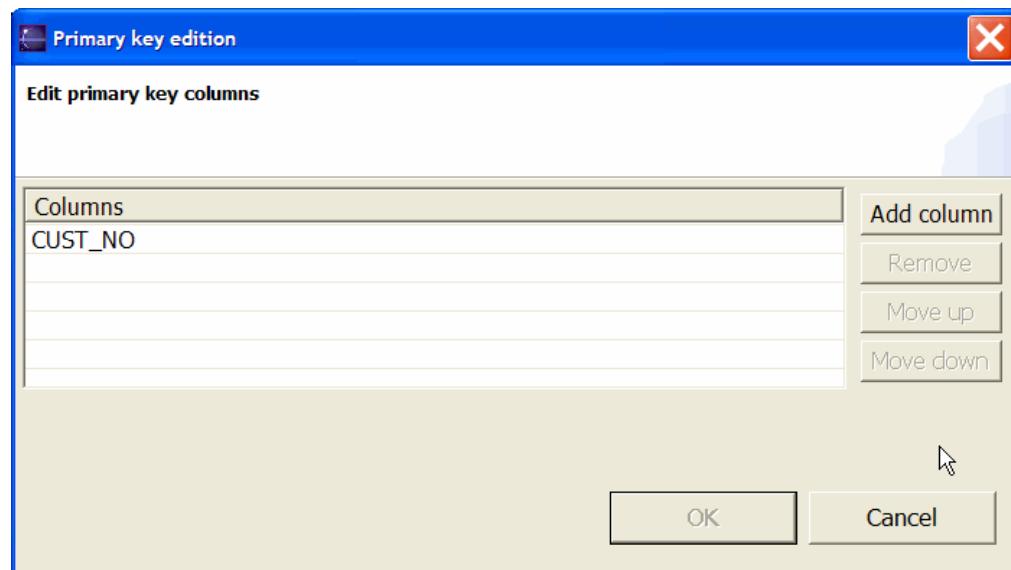


Select a Database Primary Key, right-click and select :

Properties

or

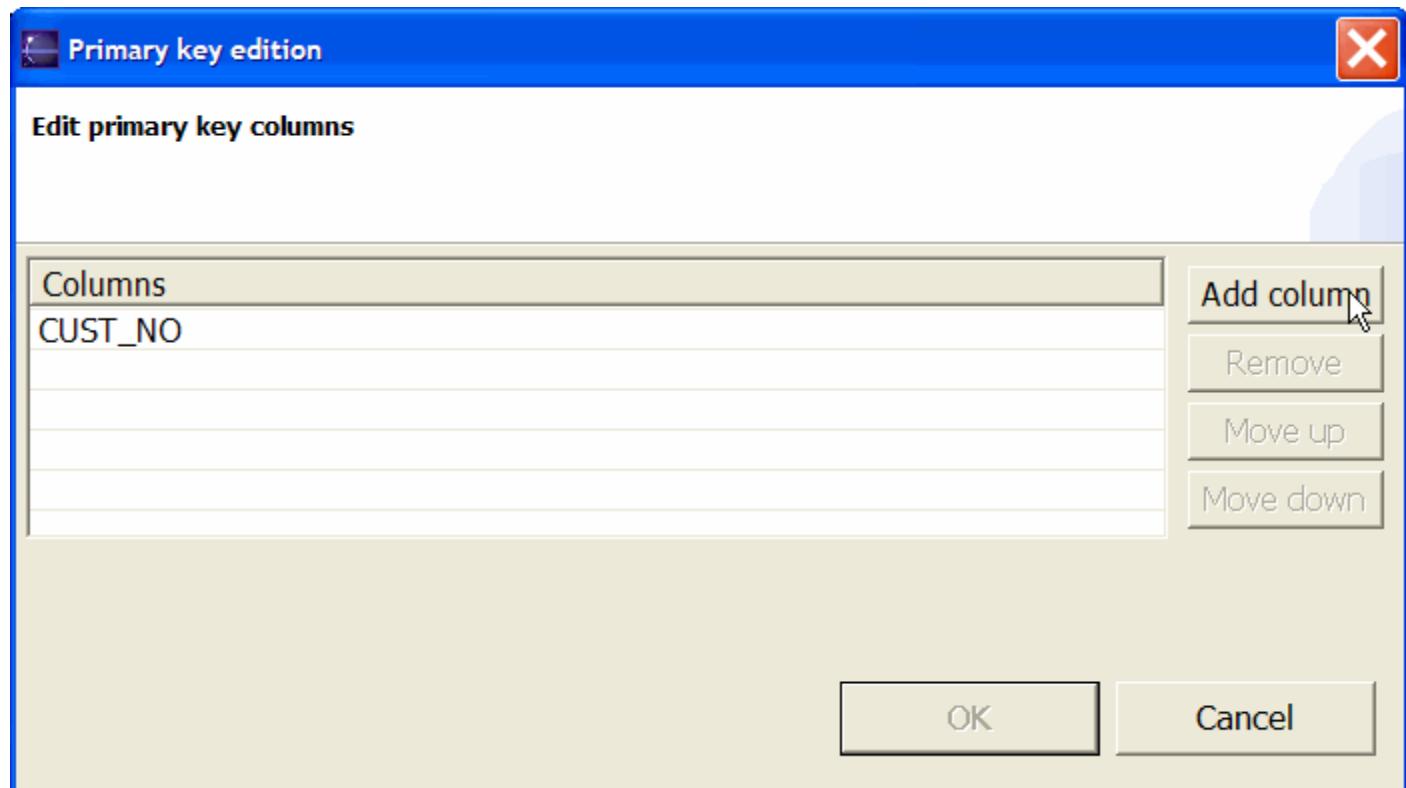
Double Click a Database Primary Key :



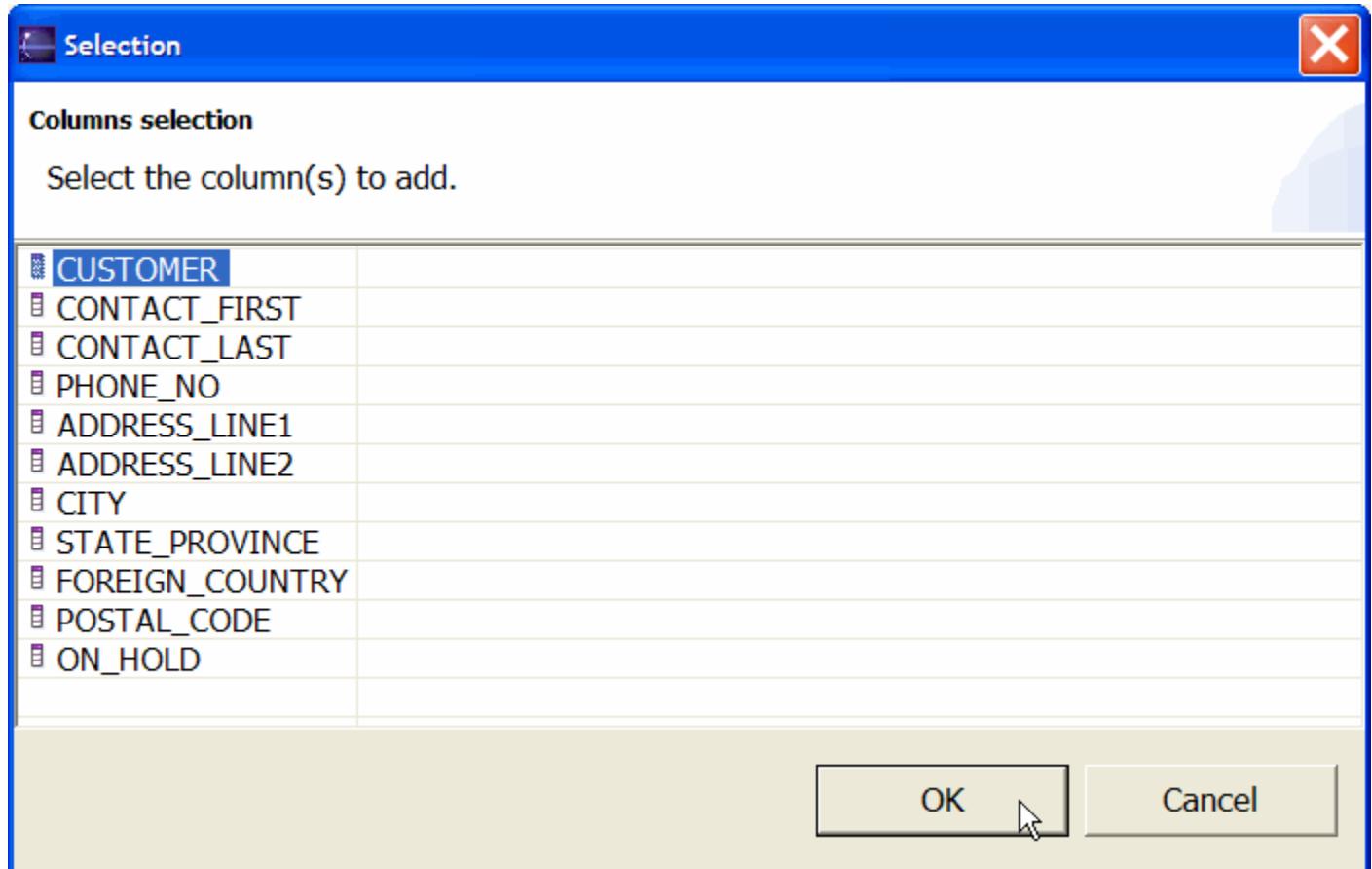
2.1. Properties

Database Primary Key properties.

2.1.1. Add Column

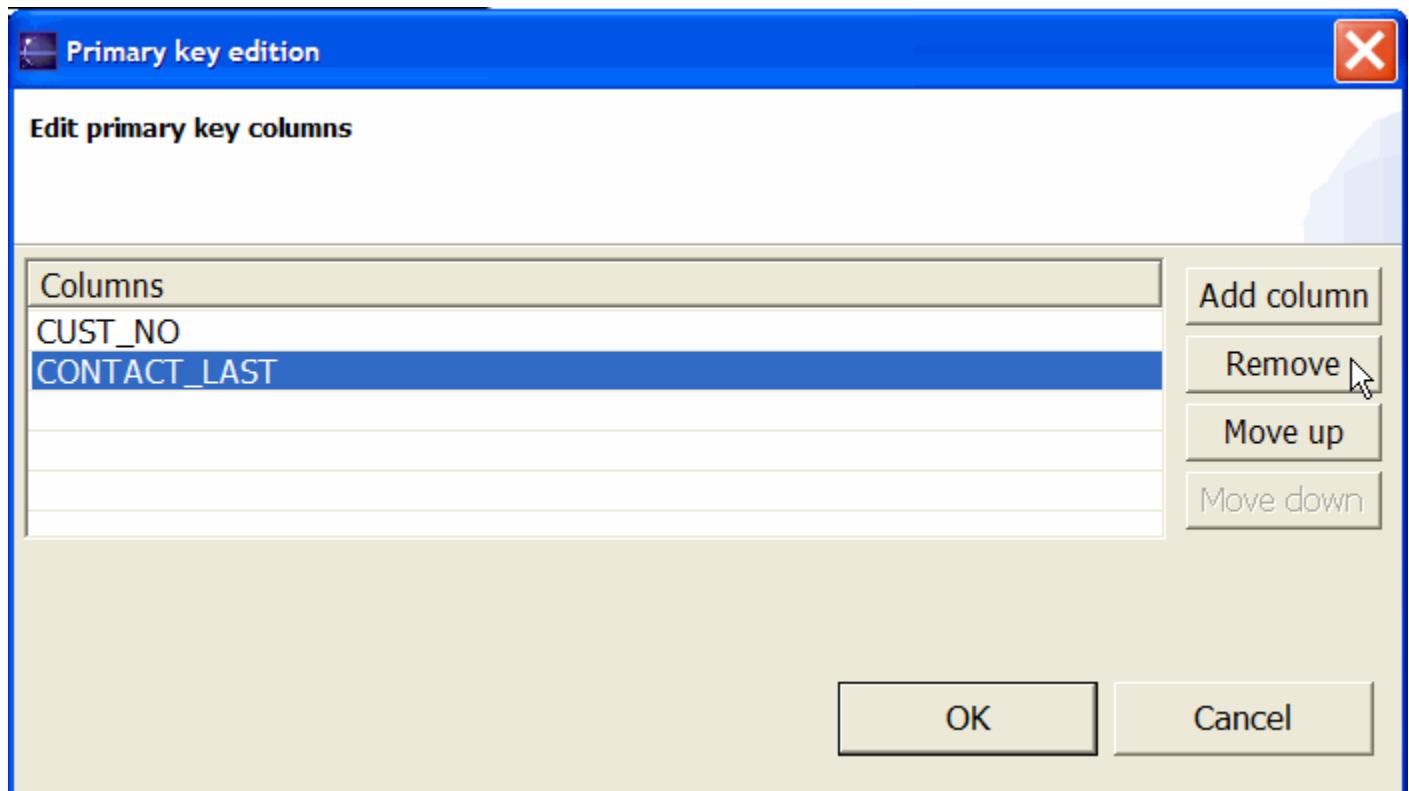


Use the **Add Column** button.



Select the needed Database Column to be part of your Database Primary Key.

2.1.2. Remove



Select a Database Column.

Then use the **Remove** button.

The selected Database Column will be removed from your Database Primary Key.

2.1.3. Move Up

You can change the Database Primary Key Column sequence inside your Database Primary Key definition.

The **Move Up** button moves up the selected Database Column.

2.1.4. Move Down

You can change the Database Primary Key Column sequence inside your Database Primary Key definition.

The **Move Down** button moves down the selected Database Column.

2.2. OK

Once you have decided to update your Database Primary Key, click the **OK** button.

The update will be done.

A detailed output will be displayed in the [DatabaseConsole](#).

The screenshot shows the DatabaseConsole interface. At the top, there's a toolbar with various icons. Below it is a main window containing a table diagram labeled "customer". The "Primary Key" section is expanded, showing two columns: "CUST_NO" and "CONTACT_FIRST". The "CONTACT_FIRST" column is highlighted with a blue selection bar. Below the primary key, a list of columns with their data types is provided:

1	CUST_NO
2	CUSTOMER
3	CONTACT_FIRST
4	CONTACT_LAST
5	PHONE_NO
6	ADDRESS_LINE1
7	ADDRESS_LINE2
8	CITY
9	STATE_PROVINCE
10	FOREIGN_COUNTRY
11	POSTAL_CODE
12	ON_HOLD

At the bottom of the main window, there's a "Properties" tab and a "Database Console" tab. The "Database Console" tab is active, displaying a log of database operations:

```

2004-10-06 10:02:11 Table customer->Foreign Key Reference->Table customer->Primary Key->Column CUST_NO
2004-10-06 10:02:11 Table sales->Foreign Key->Table customer->Primary Key->Column CUST_NO
2004-10-06 10:02:11 Table customer->Primary Key->Column CUST_NO : Updated
2004-10-06 10:02:11 Table customer->Foreign Key Reference sales->Table customer->Primary Key->Column CONTACT_FIRST
2004-10-06 10:02:12 Table sales->Foreign Key customer->Table customer->Primary Key->Column CONTACT_FIRST
2004-10-06 10:02:12 Table customer->Primary Key : Sorted
2004-10-06 10:02:12 Table customer->Primary Key->Column CONTACT_FIRST : Added

```

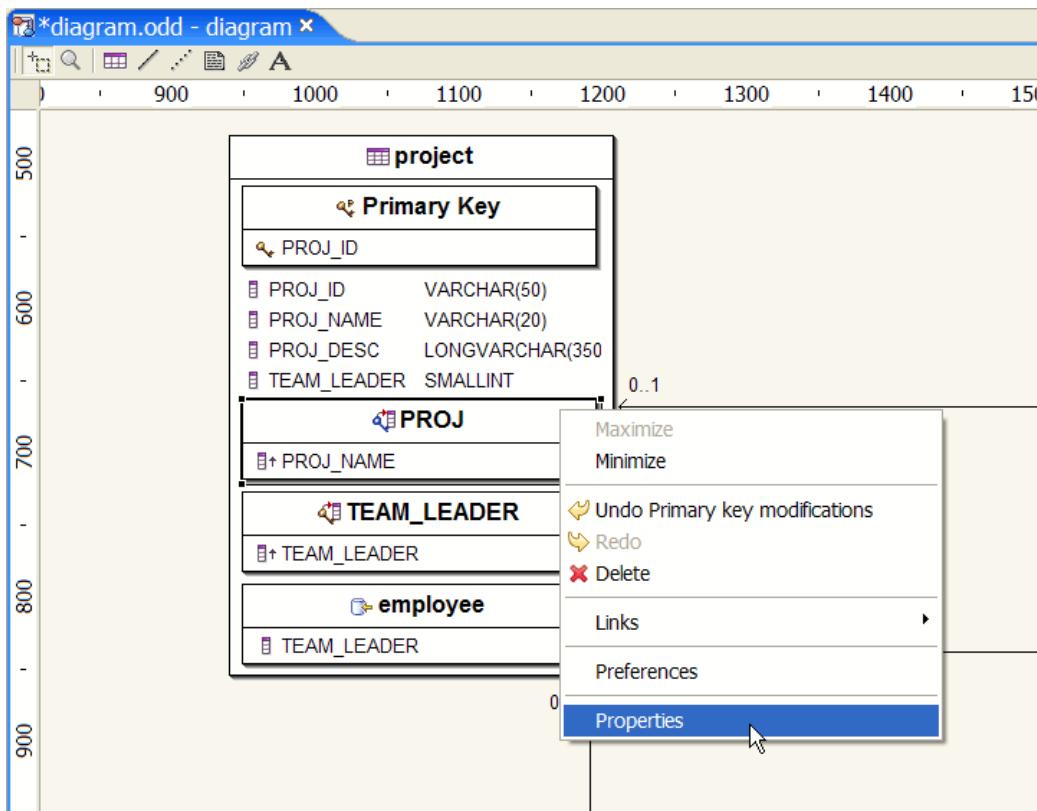
Database Editor Properties – Index

1. Introduction
2. Index
 1. Properties
 1. Name
 2. Unique
 3. Add Column
 4. Remove
 5. Move Up
 6. Move Down
 2. OK

1. Introduction

In this section you will learn how to use Database Editor Index properties.

2. Index

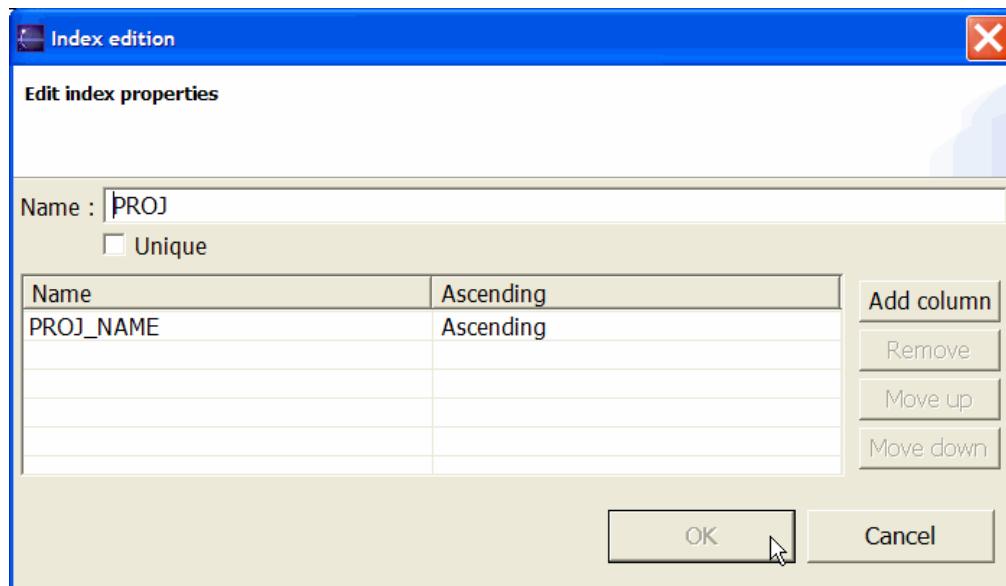


Select a Database Index, right-click and select :

Properties

or

Double Click a Database Index :



2.1. Properties

Database Index properties.

2.1.1. Name

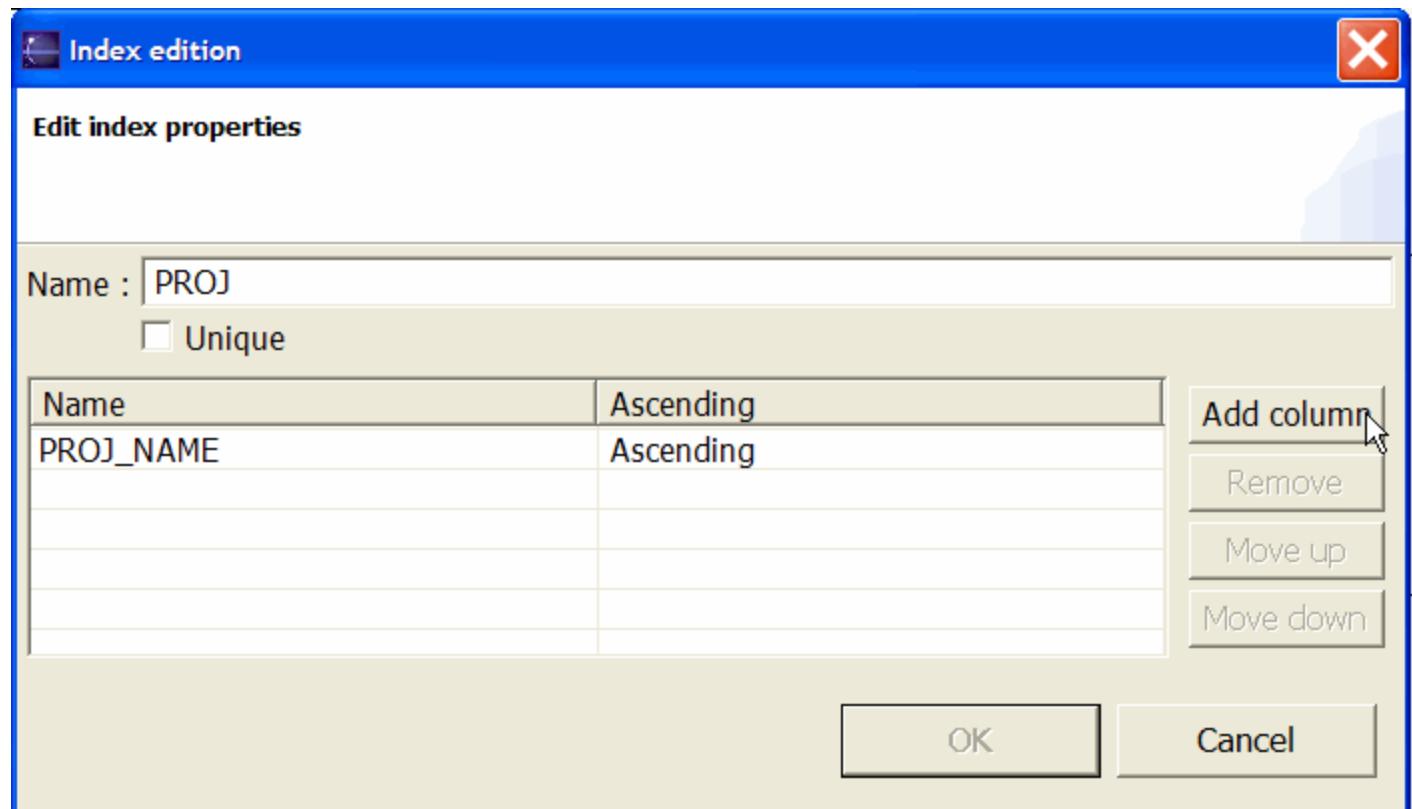
Index Name.

This name should be unique among your Database Indexes in your current Database Table.

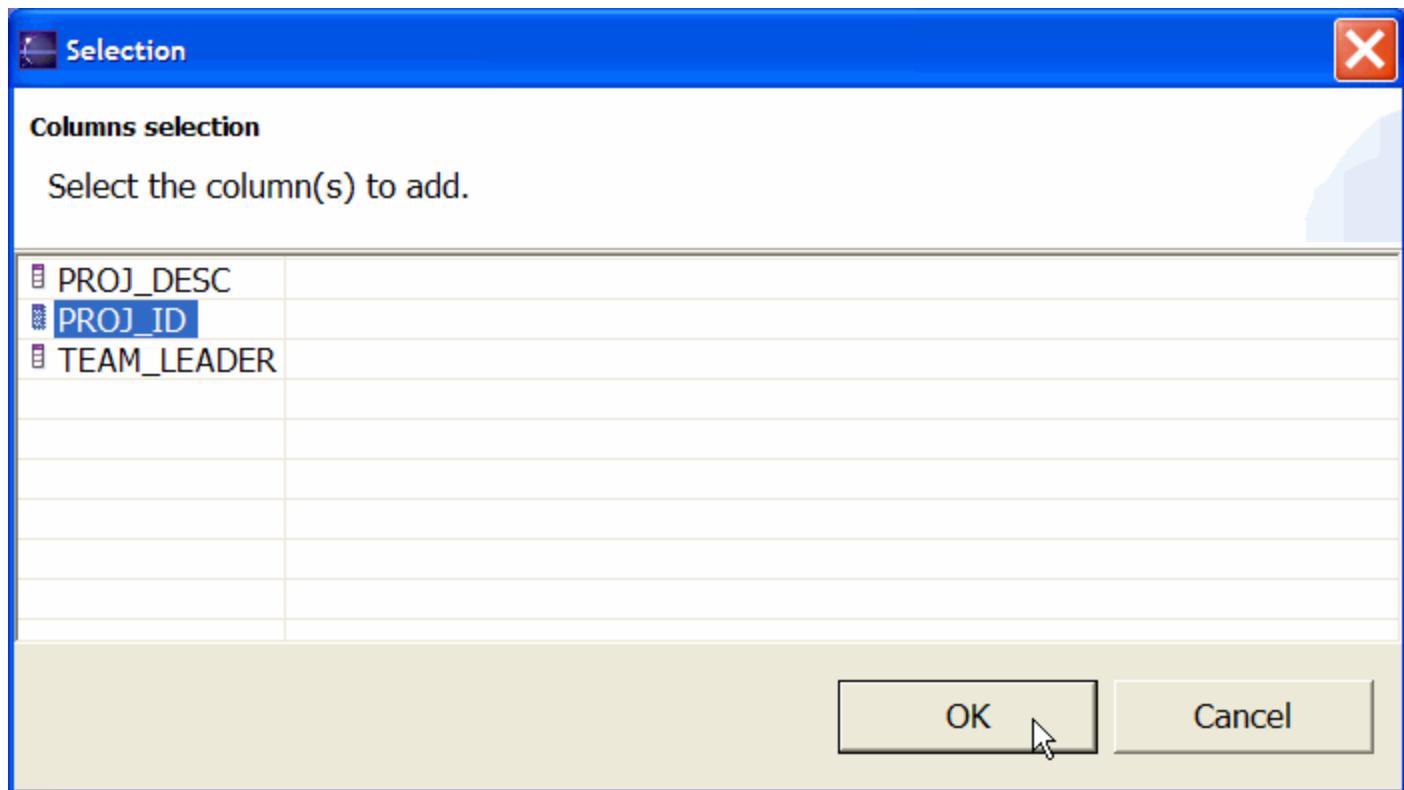
2.1.2. Unique

A Database Index could contain unique tokens or not.

2.1.3. Add Column

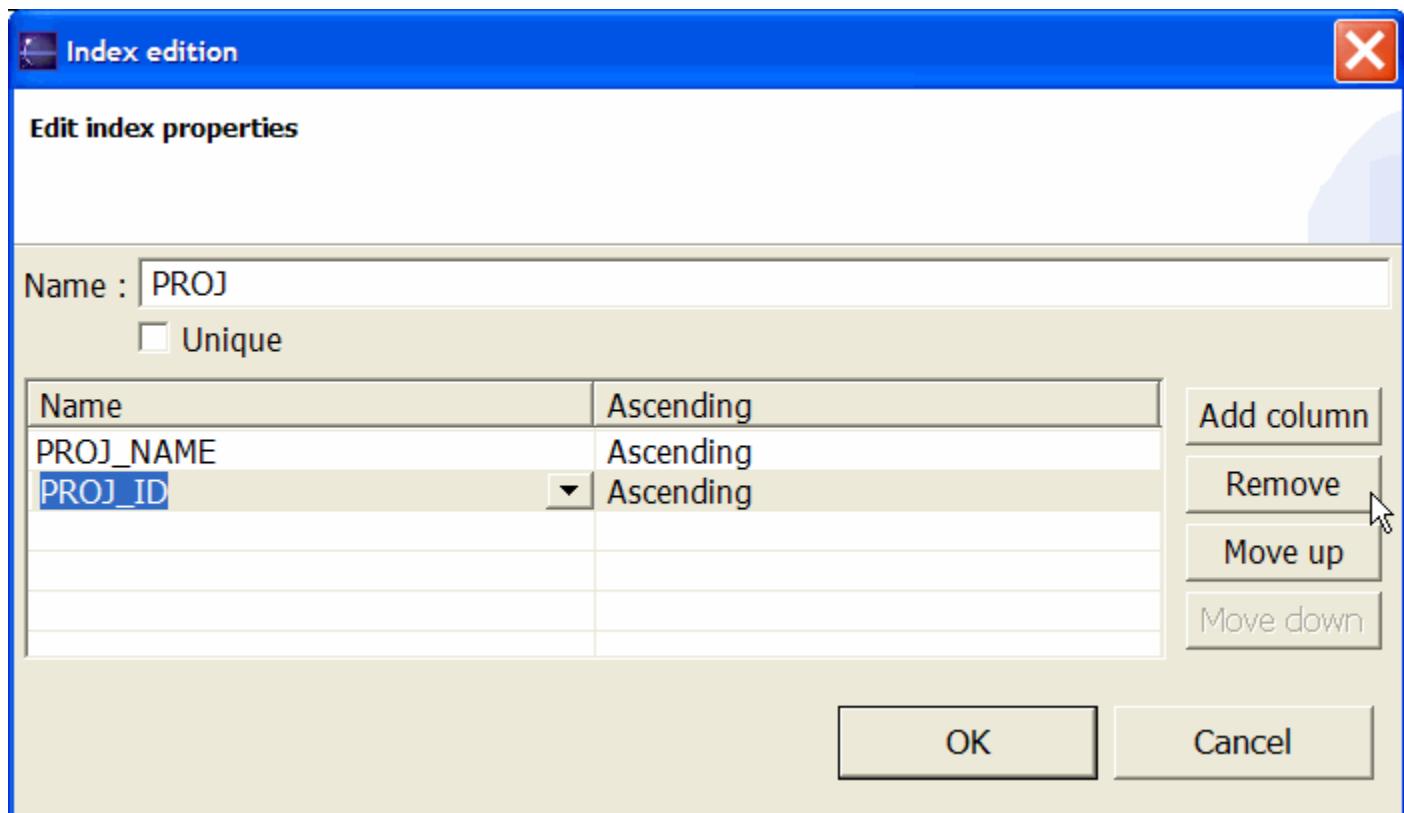


Use the **Add Column** button.



Select the needed Database Column to be part of your current Database Index.

2.1.4. Remove



Select a Database Column.

Then use the **Remove** button.

The selected Database Column will be removed from your current Database Index.

2.1.5. Move Up

You can change the Database Index Column sequence inside your current Database Index definition. The **Move Up** button moves up the selected Database Column.

2.1.6. Move Down

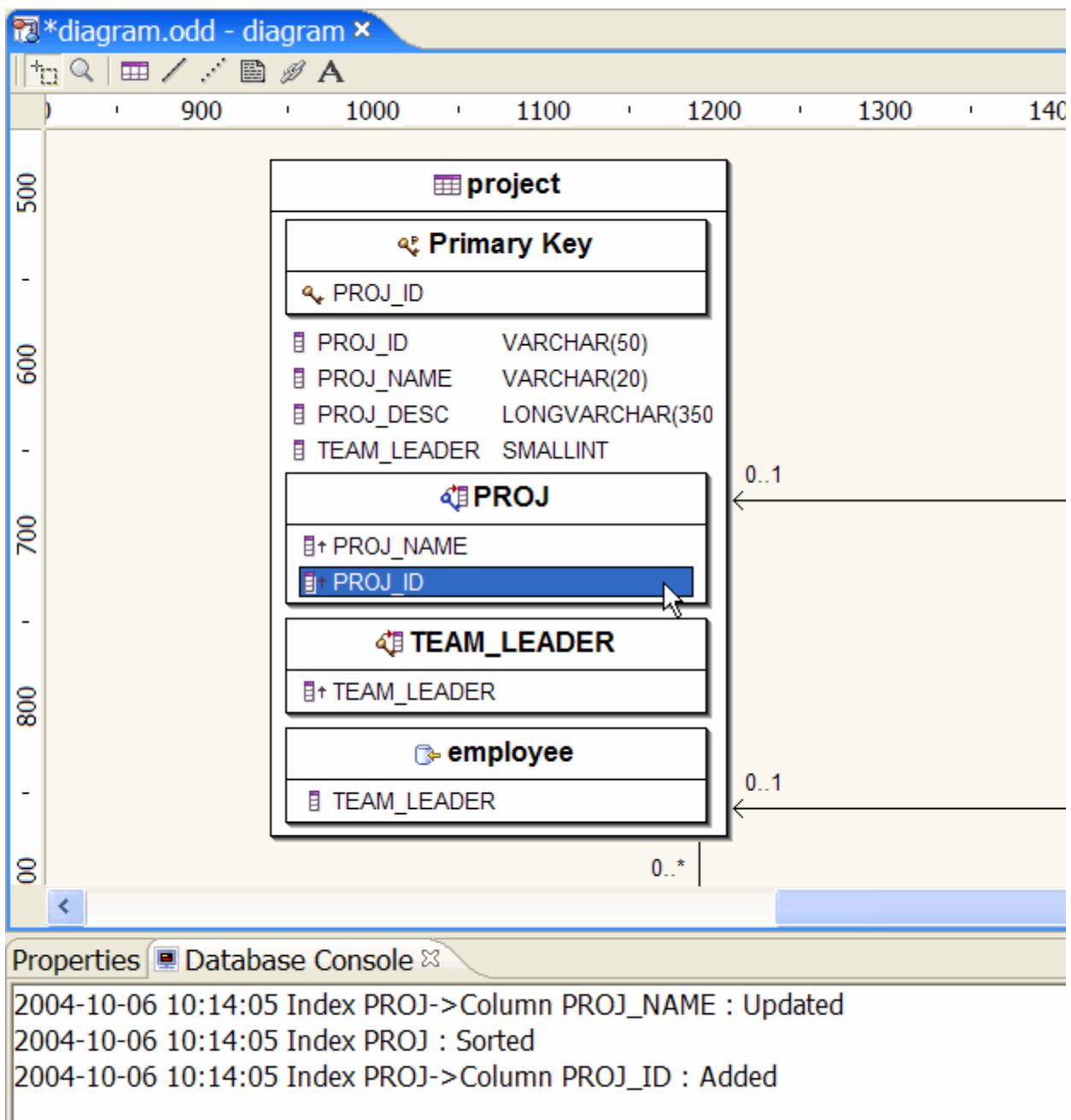
You can change the Database Index Column sequence inside your current Database Index definition. The **Move Down** button moves down the selected Database Column.

2.2. OK

Once you have decided to update your Database Index, click the **OK** button.

The update will be done.

A detailed output will be displayed in the [DatabaseConsole](#).



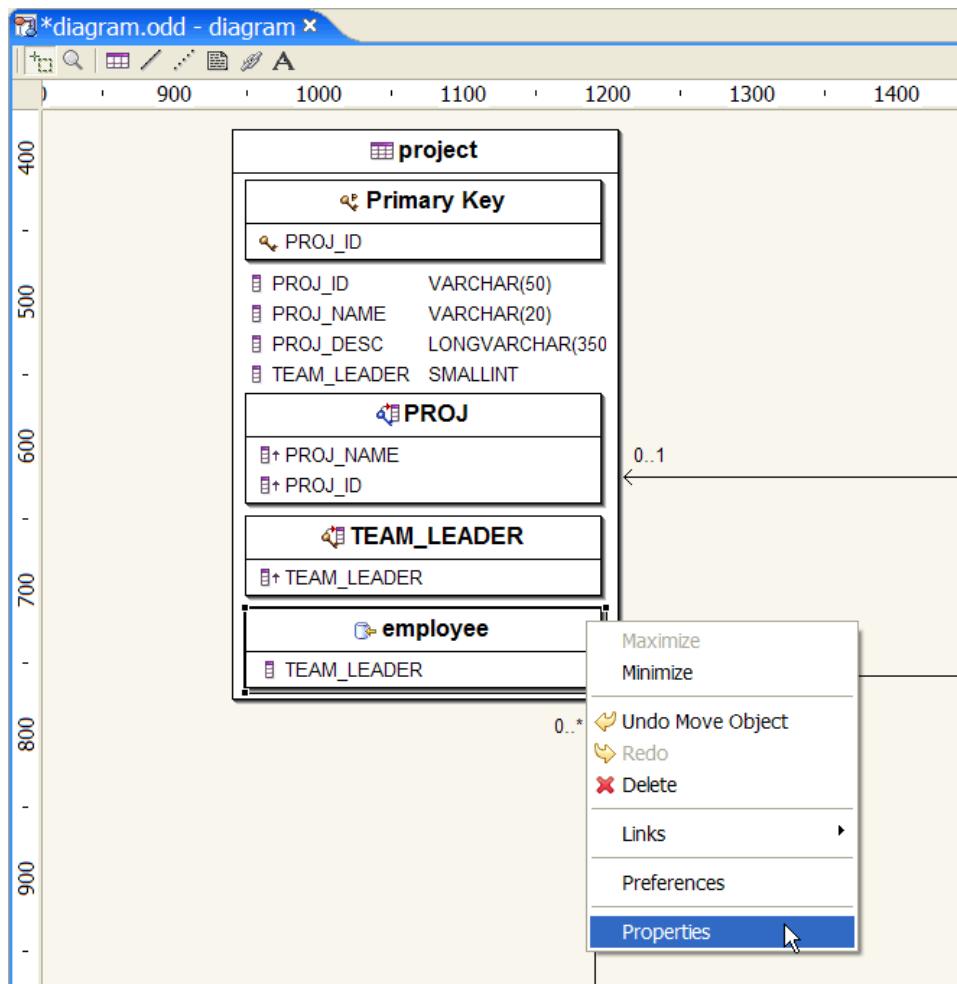
Database Editor Properties – Foreign Key

1. Introduction
2. Foreign Key
 1. Properties
 1. Primary Key Table
 2. Foreign Table
 3. Primary Cardinality
 4. Foreign Cardinality
 5. onUpdate
 6. onDelete
 7. Add Column
 8. Remove
 2. OK

1. Introduction

In this section you will learn how to use Database Editor Index properties.

2. Foreign Key



Select a Database Foreign Key, right-click and select :

Properties.

or

Double Click a Database Foreign Key :

or

Double Click the corresponding association :

2.1. Properties

Database Index properties.

2.1.1. Primary Key Table

Foreign Database Primary Key Table name.

2.1.2. Foreign Table

Current Database Table name.

2.1.3. Primary Cardinality

The Primary Key exists or does not exist.

This value is immutable.

2.1.4. Foreign Cardinality

If the Primary Key exists, the foreign could reference the Primary Key.

This value is mutable.

2.1.5. OnUpdate

On Update Foreign Key behaviour.

This value is inherited from the [Database Schema](#) properties.

2.1.6. OnDelete

2.1.6. ON DELETE

This value is inherited from the [Database Schema properties](#).

2.1.7. Add Column

Edit foreign key properties

⚠ Unmapped columns

Primary key table :	employee
Foreign table :	project
Primary Cardinality :	0..1
Foreign Cardinality :	0..*
onUpdate :	none
onDelete :	none

Foreign columns Primary columns

Foreign columns	Primary columns

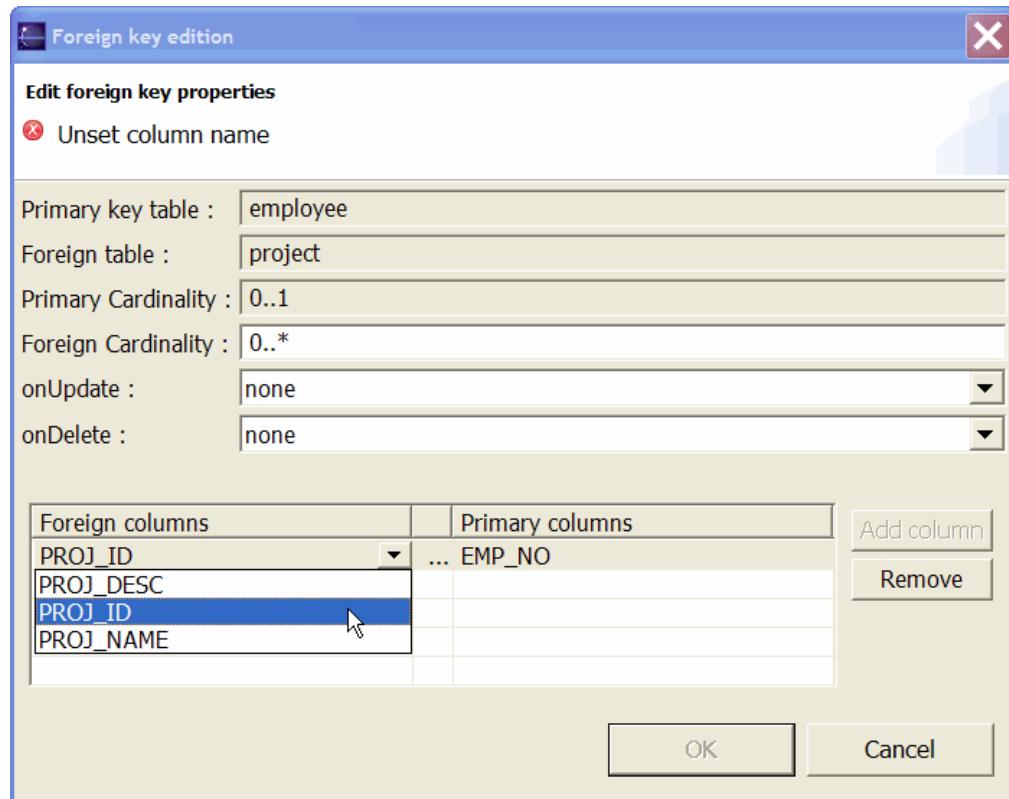
Add column Remove

OK Cancel

Use the **Add Column** button.

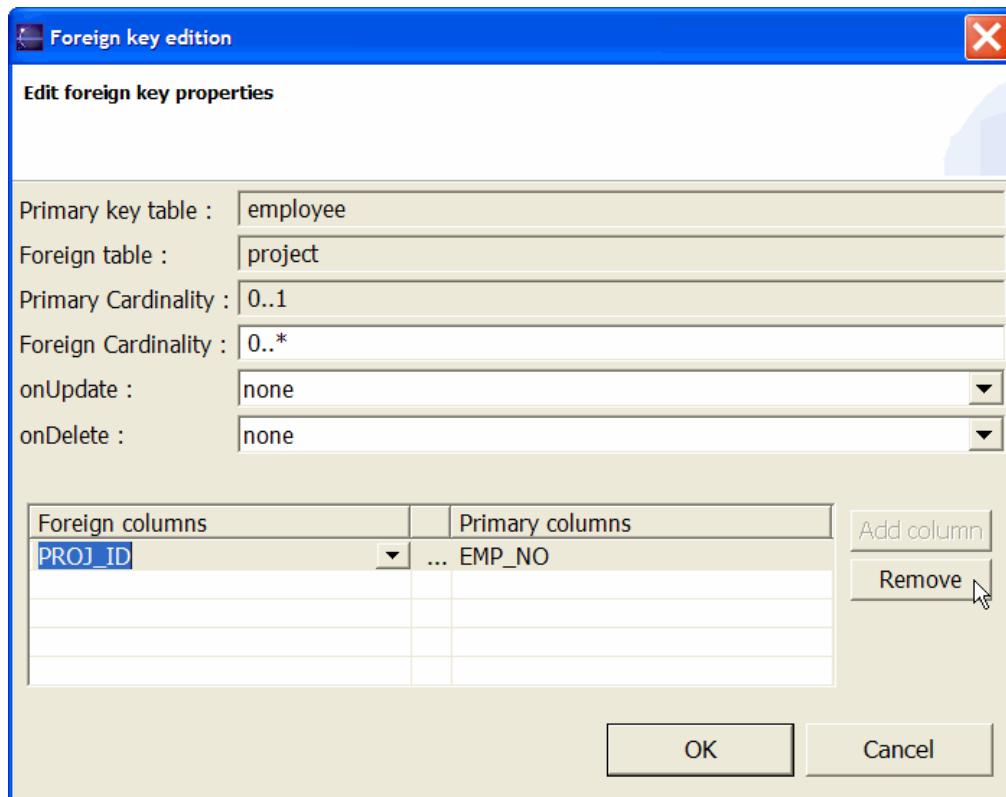
The screenshot shows a software interface titled "Selection" with a blue header bar. In the top right corner is a red "X" button. The main area is titled "Columns selection" and contains the instruction "Select the column(s) to add." Below this is a table with one visible row. The first column of the table has a small icon and the text "EMP_NO". At the bottom of the window are two buttons: "OK" and "Cancel".

Select the needed Foreign Database Primary Column to be part of your current Database Foreign Key definition.



The Foreign Primary key Column is added to your Foreign Key Columns definition array. Then select the local Database Column mapped to the Foreign Primary Key Column.

2.1.8. Remove



Select a Database Column.

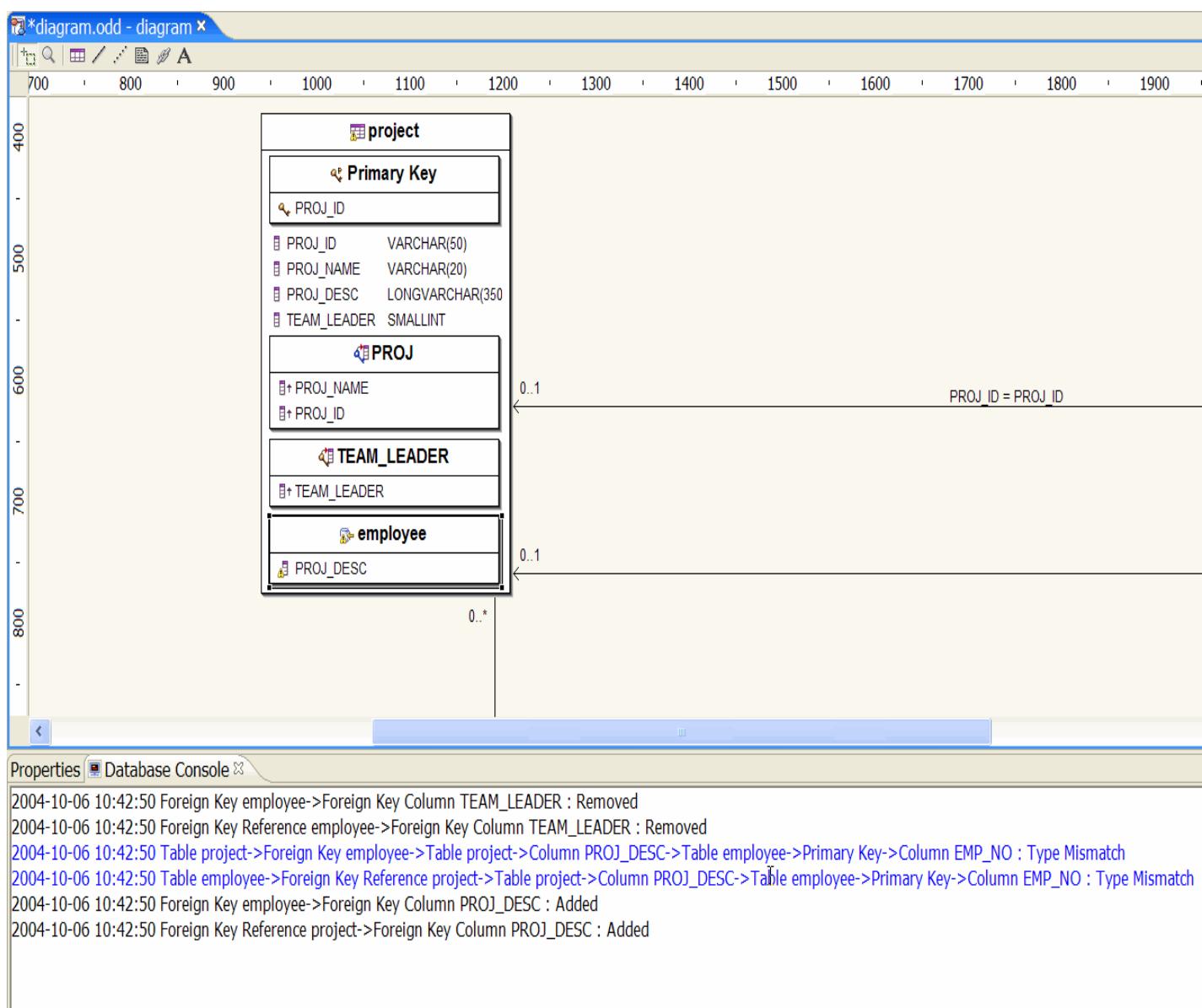
Then use the **Remove** button.

The selected Database Column will be removed from your current Database Foreign Key.

2.2. OK

Once you have decided to update your Database Foreign Key, click the **OK** button.
The update will be done.

A detailed output will be displayed in the [DatabaseConsole](#).



Code Generators

1. [Introduction](#)
2. [Torque Mapping Object Resources](#)
3. [Object Relational Bridge Resources](#)
4. [Hibernate](#)

1. Introduction

You will learn in this chapter how to generate Java code to access your Database.

EclipseDatabase proposes several wizards who will help you to generate your Java code.

- [Torque Mapping Object Resources](#)
- [Object Relational Bridge Resources](#)
- [Hibernate Configuration and Mapping Resources](#)
- [Hibernate Java Resources](#)

Generalities about the way EclipseDatabase manage the wizards have been explained in the [Modeling in the Workspace Introduction](#) chapter.

Torque Mapping Object Ressources

1. [Introduction](#)
2. [Torque Mapping Object Resources](#)
 1. [Source Folder](#)
 2. [Package](#)
 3. [Schema](#)
 4. [Options](#)
 1. [Base Prefix](#)
 2. [Add GetByNameMethod](#)
 3. [Add Intake Retrievable](#)
 4. [Add Save Method](#)
 5. [Add Timestamp](#)
 6. [Complex Object Model](#)
 7. [Use Managers](#)
 8. [Generate Deprecated](#)
 5. [Finish](#)
3. [Templates](#)
4. [Additional Resources](#)

1. Introduction

The purpose of this chapter is to show how to generate Torque Object Model resources.

This wizard will generate :

- `Torque.properties` file
- Torque Object Model Java classes

This wizard needs to use a target Java Project.

EclipseDatabase provides the current embedded [Torque Generator Project Documentation](#).

2. Torque Mapping Object Resources

To start the Torque Mapping Object Resources wizard, select :

File->New->Other->Database->Torque Mapping Object Resources

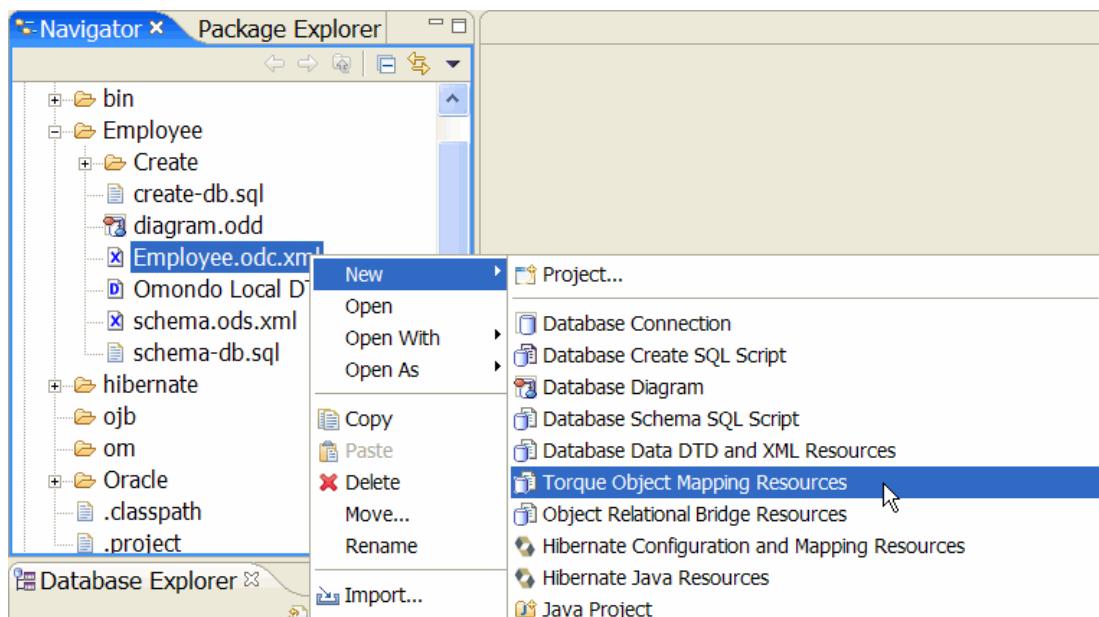
or

Select an object in the Package Explorer View or the Navigator View then right-click and select :

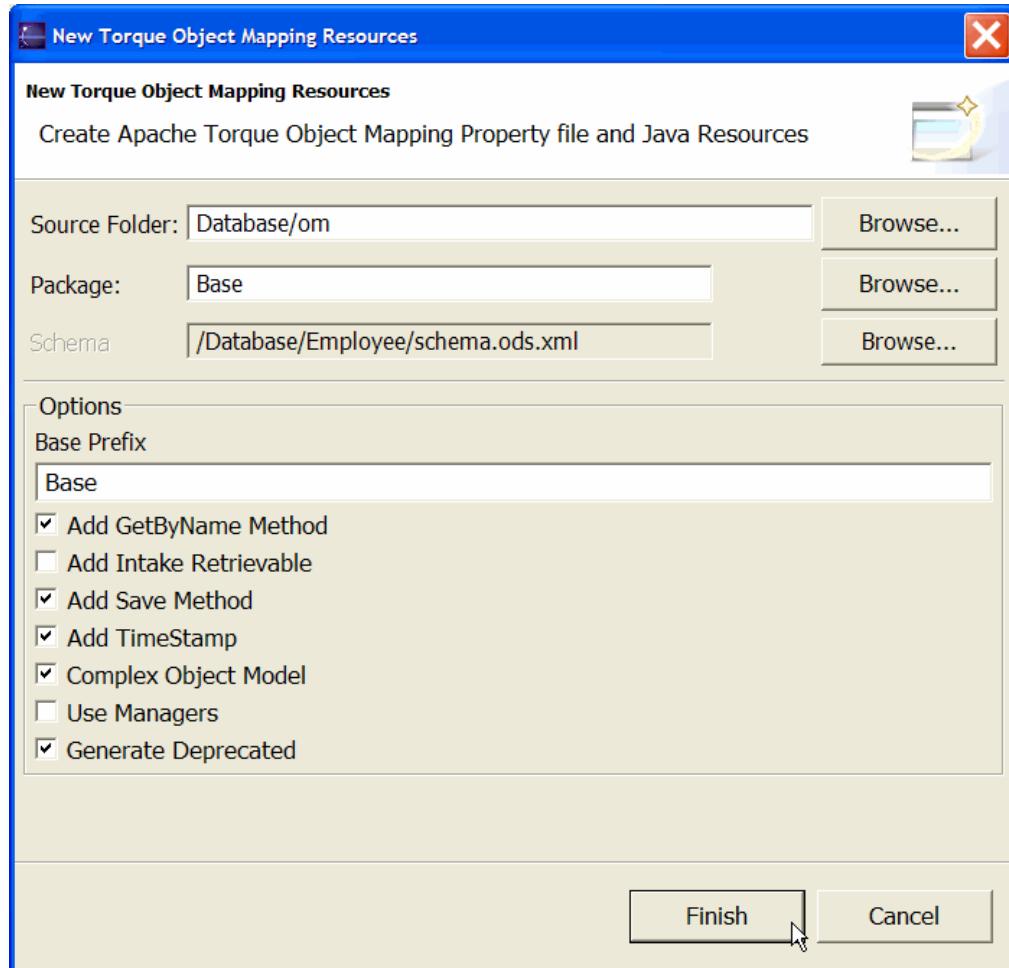
New->Other->Database->Torque Mapping Object Resources

The selection is contextual.

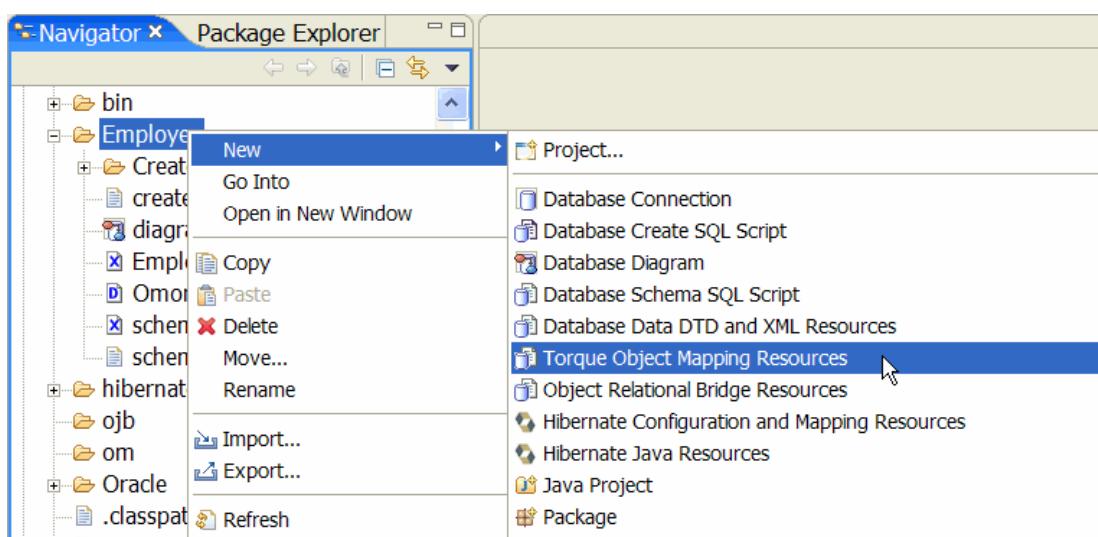
This means that if you select an existing Database Object.



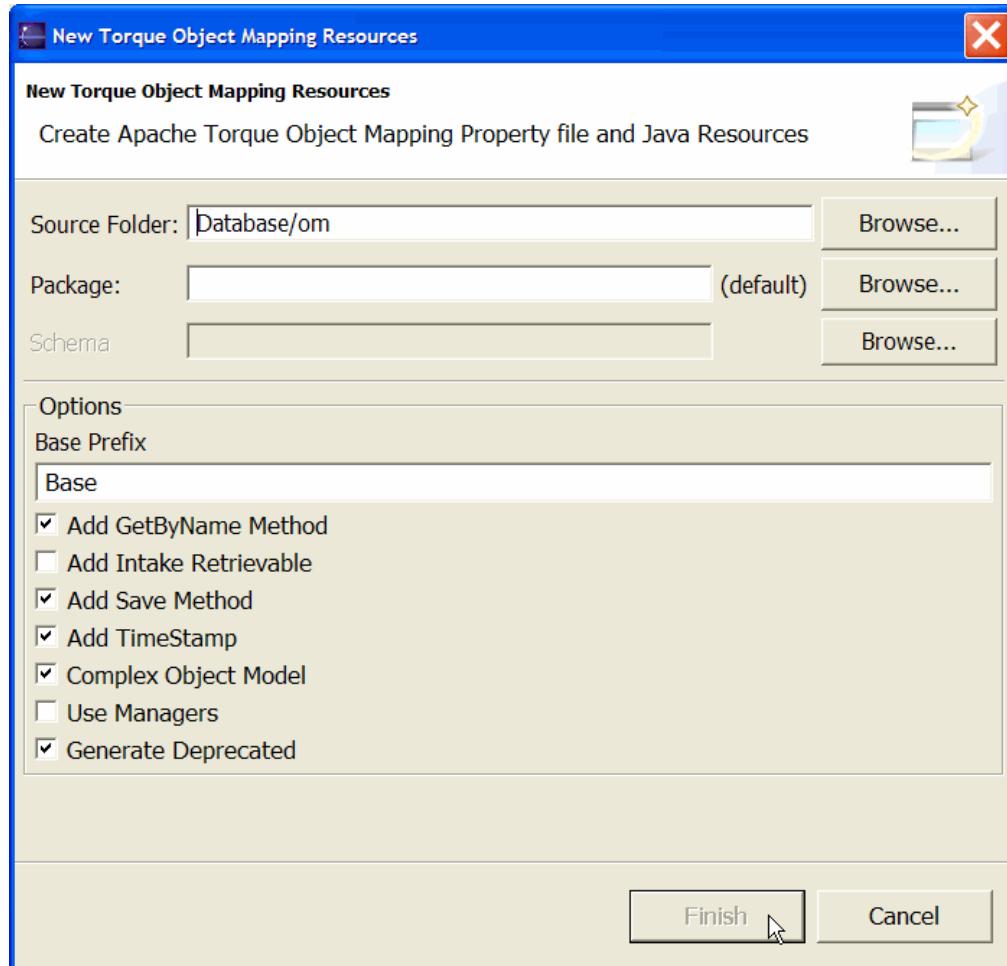
The wizard will be opened with a selected Database Connection if applicable.



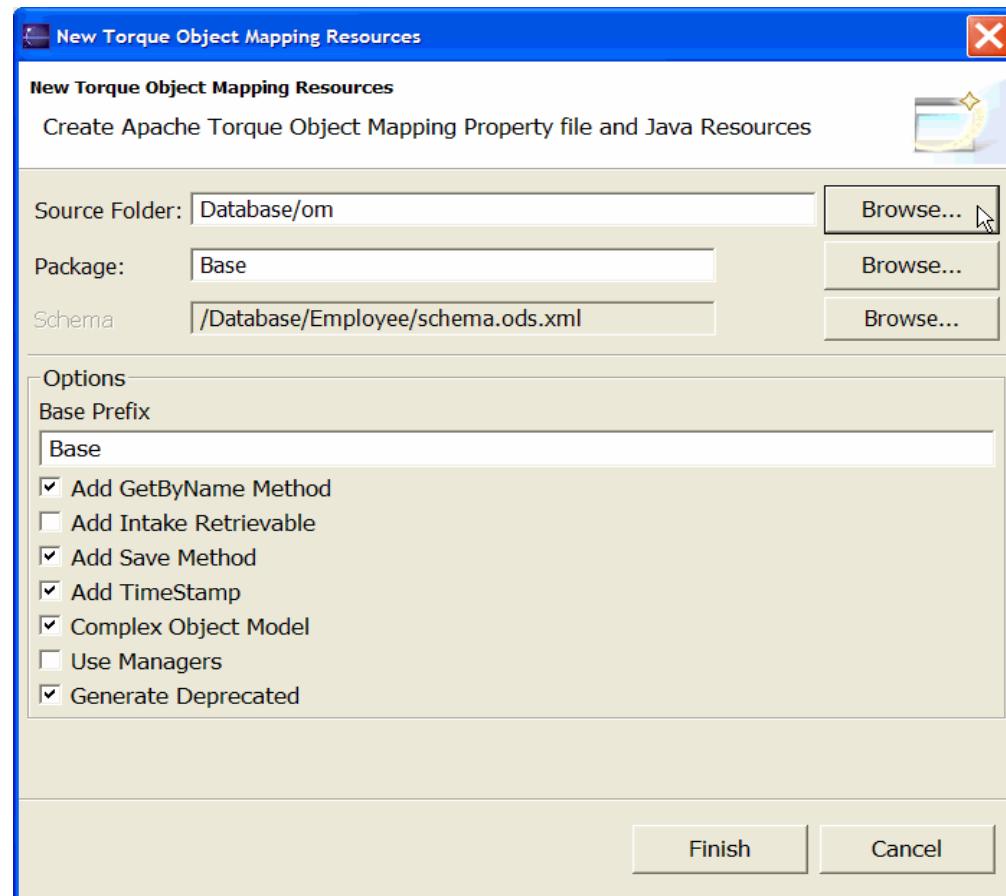
Otherwise :



The wizard will be opened without any selected Database Connection.



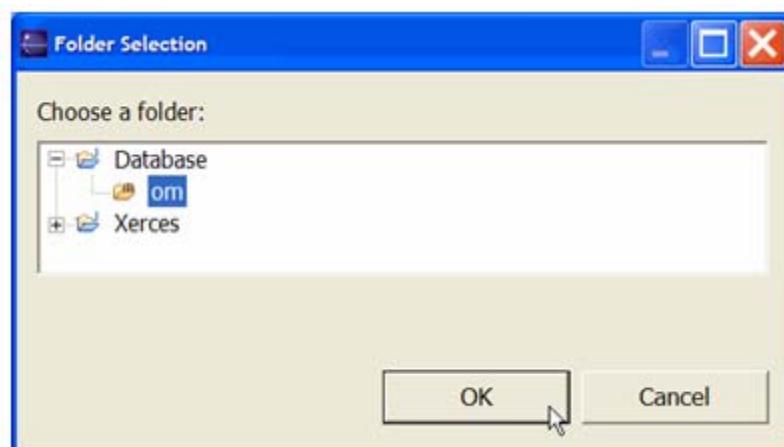
2.1. Source Folder



Type in the Java Source Folder text field the targeted Java Source Folder.

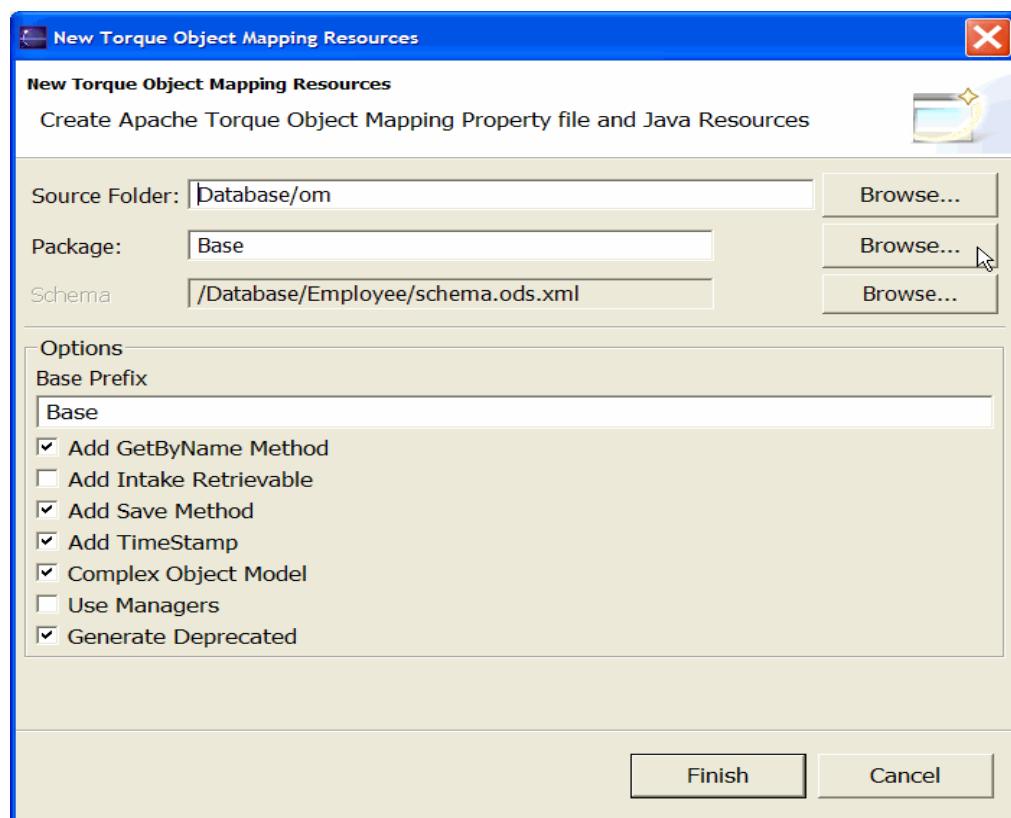
The Java Source Folder should exist.

Otherwise use the **Browse** button :



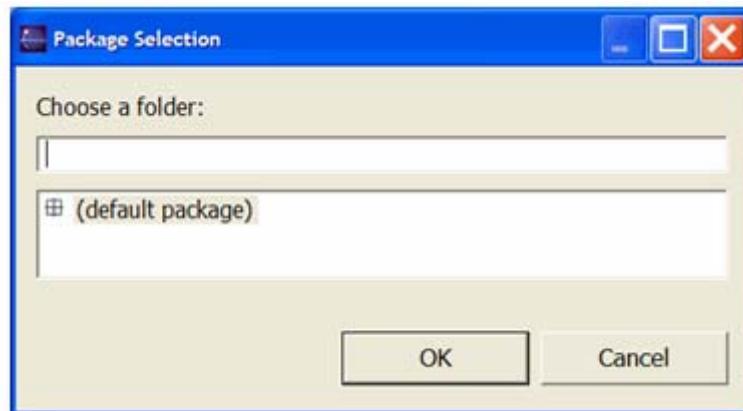
Select the appropriate Java Source Folder.

2.2. Package



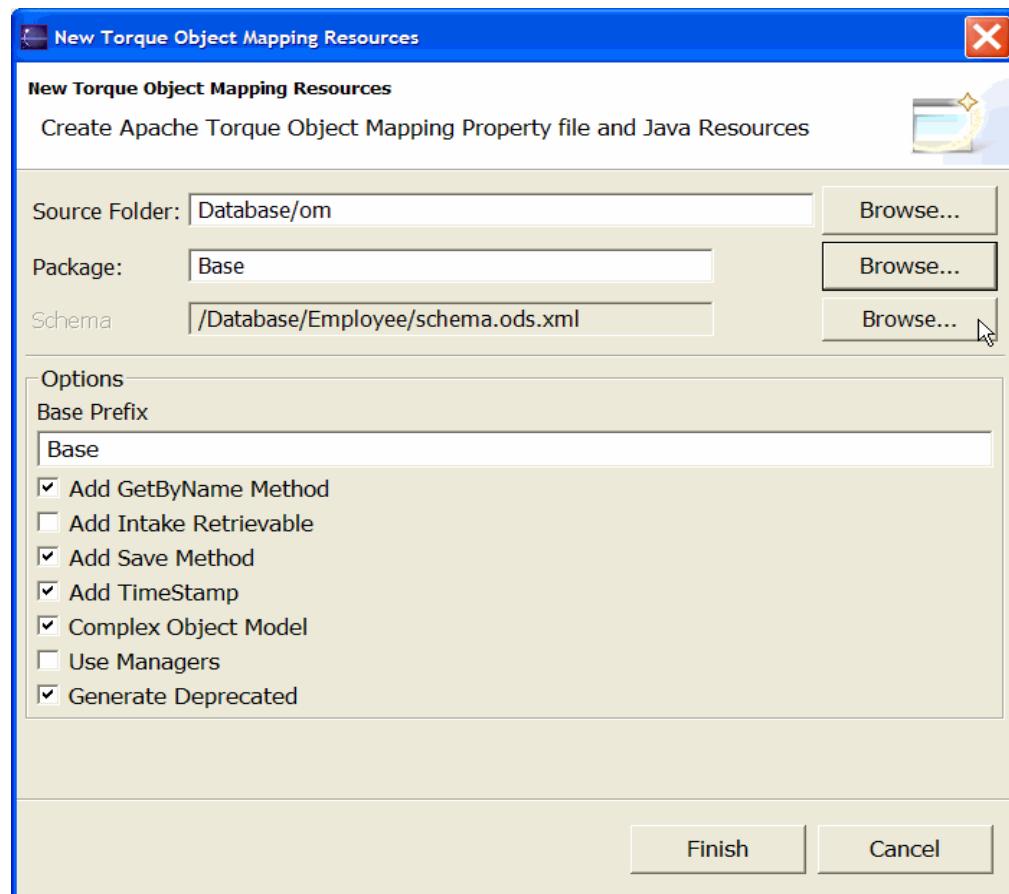
Type in the Package text field the targeted Java Package.

Otherwise use the **Browse** button :

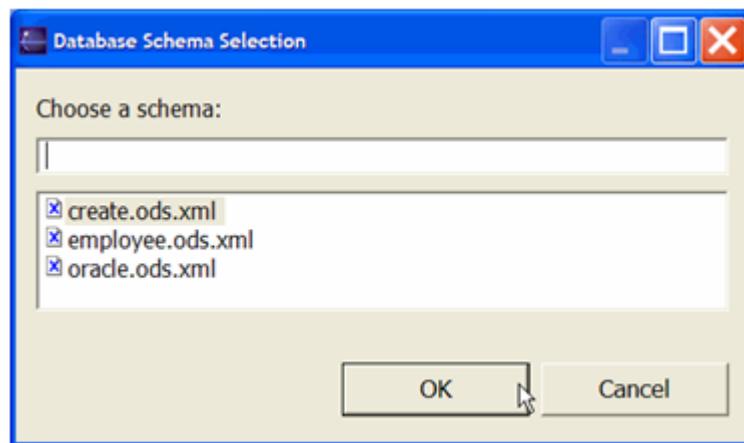


Select the appropriate targeted Java Package.

2.3. Schema



Use the **Browse** button :



The Database Schema selector displays the available Database Schema in your current Project.

Select the appropriate Database Schema.

2.4. Options

You can set various Torque options.

For further details about Torque options, you will find a detailed page [here](#).

2.4.1. Base Prefix

Inherited Database Schema Torque Base Class.

2.4.2. Add GetByNameMethod

If checked, Torque adds methods to get database fields by name/position.

2.4.3. Add Intake Retrievable

If checked, the data objects will implement Intake's Retrievable interface

2.4.4. Add Save Method

If checked, Torque adds tracking code to determine how to save objects.

2.4.5. Add Timestamp

If checked Torque puts time stamps in generated Object Model files.

2.4.6. Complex Object Model

If checked, Torque generates data objects with collection support and methods to easily retrieve foreign key relationships.

2.4.7. Use Managers

If checked, Torque will generate Manager classes that use JCS for caching.

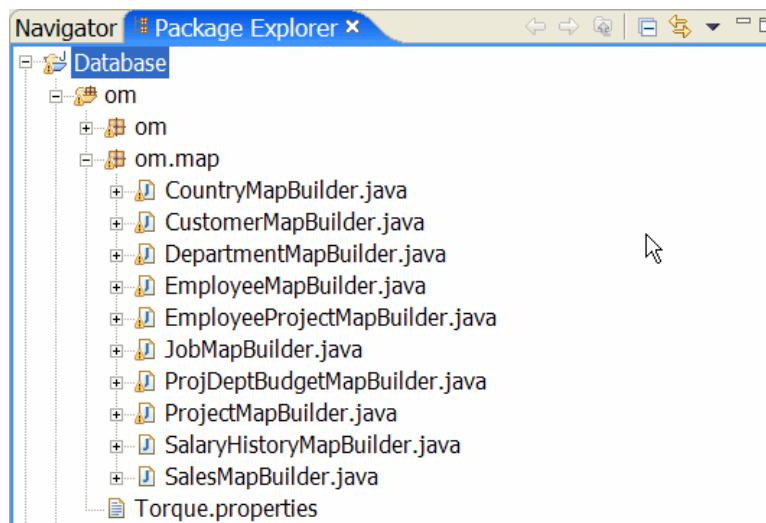
(Still considered experimental.)

2.4.8. Generate Deprecated

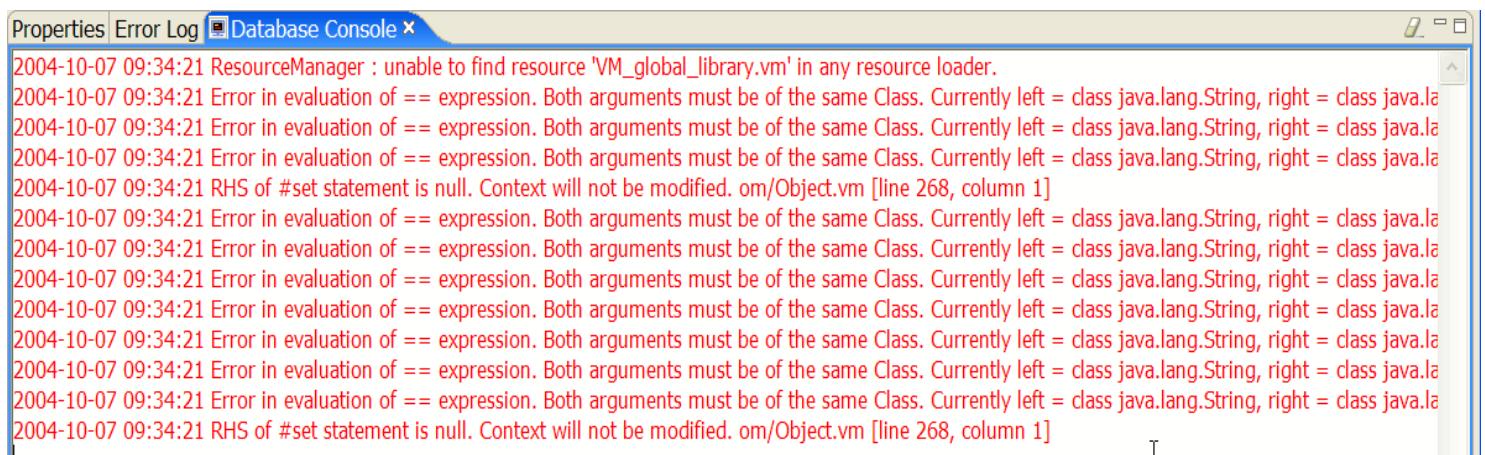
If checked, Torque will generate deprecated methods.

2.5. Finish

The code generation process will be performed in the targeted Java Source Folder and its designated target package.



Detailed information will be displayed in the DatabaseConsole window.

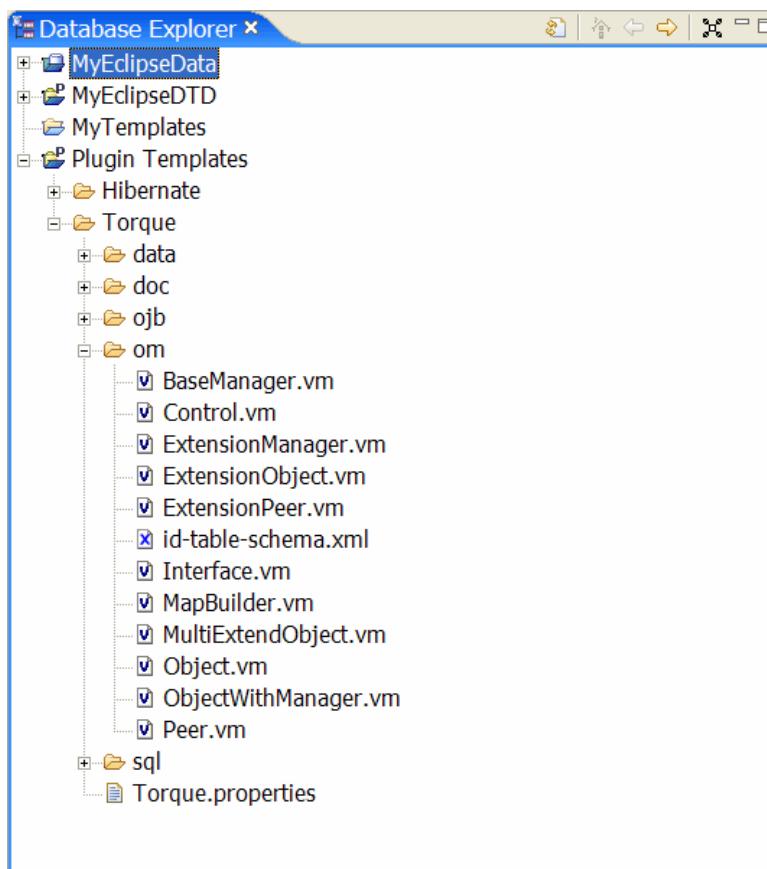


3. Templates

This wizard is [Template](#) based.

Below your [Plugin Templates](#) or [MyTemplates](#) you should see the following directory.

Torque/om



These templates are [Velocity](#) based and use the [Torque Model API](#).
The Torque.properties is provided as a convenience.

4. Additional Resources

Here are several links provided for further Apache Torque related information.

- [Apache Torque Project](#)
- [Torque Wiki](#)
- [Apache Turbine Project](#)
- [Turbine Wiki](#)
- [Turbine and Torque Mailing Lists](#)
- [Apache Velocity Project](#)
- [Velocity Mailing Lists](#)

Object Relational Bridge Ressources

1. Introduction
2. Object Relational Bridge Resources
 1. Source Folder
 2. Package
 3. Schema
 4. Finish
3. Templates
4. Additional Resources

1. Introduction

The purpose of this chapter is to show how to generate Object Relational Bridge (OJB) resources. This wizard needs to use a target Java Project.

This wizard will generate :

- Repository files
- OJB Java classes

2. Object Relational Bridge Resources

To start the Object Relational Bridge Resources wizard, select :

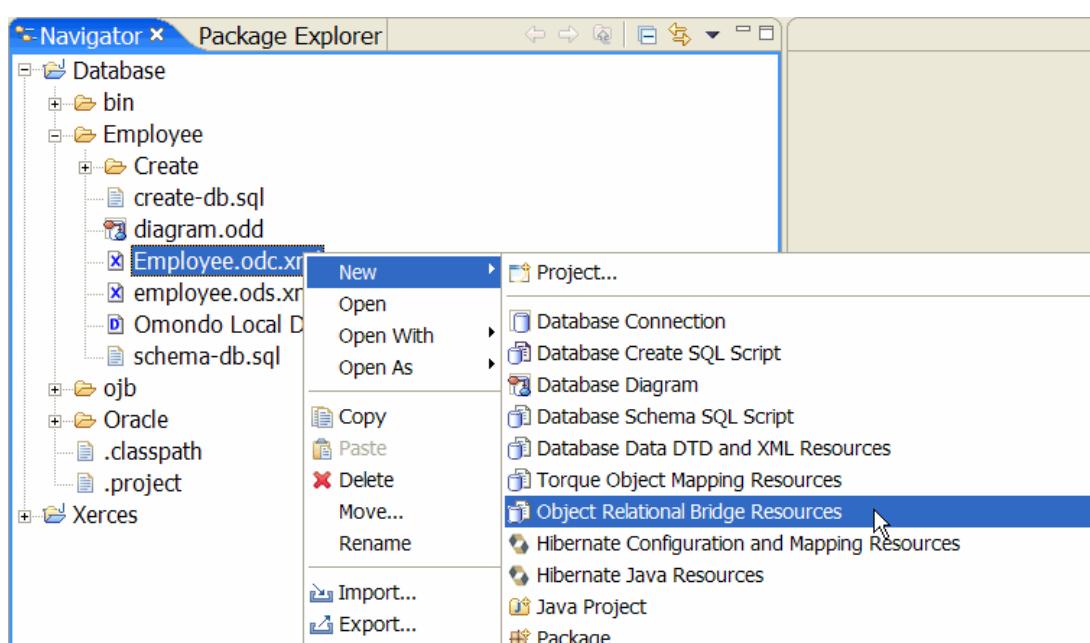
File->New->Other->Database->Object Relational Bridge Resources

or

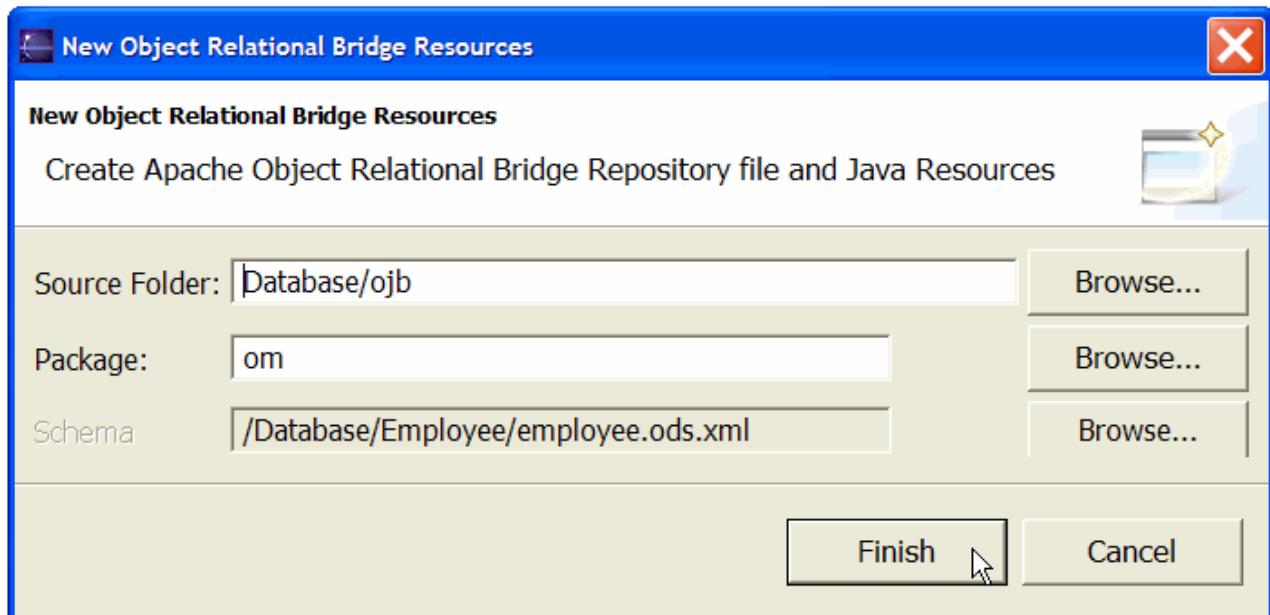
Select an object in the Package Explorer View or the Navigator View then right-click and select : New->Other->Database->Object Relational Bridge Resources

The selection is contextual.

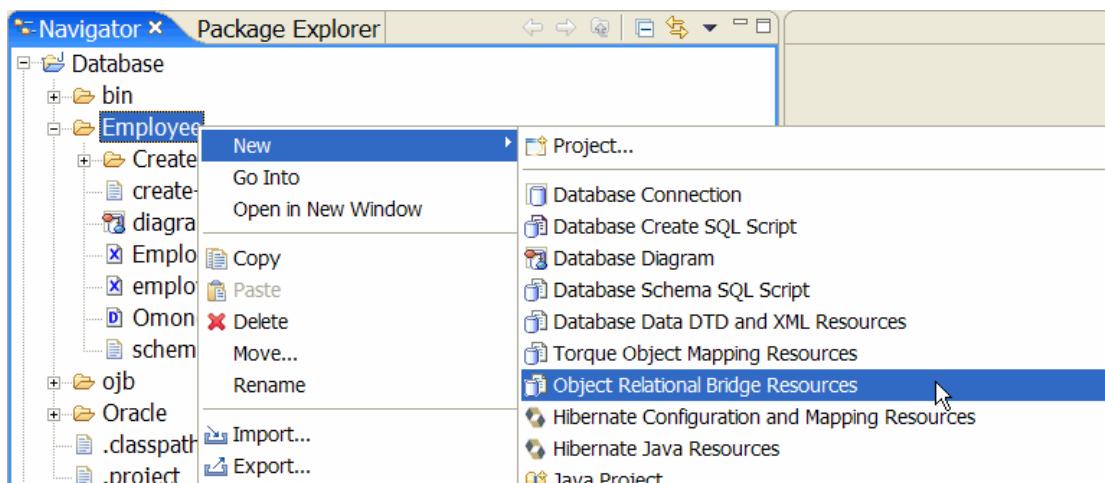
It means that if you select an existing Database Object



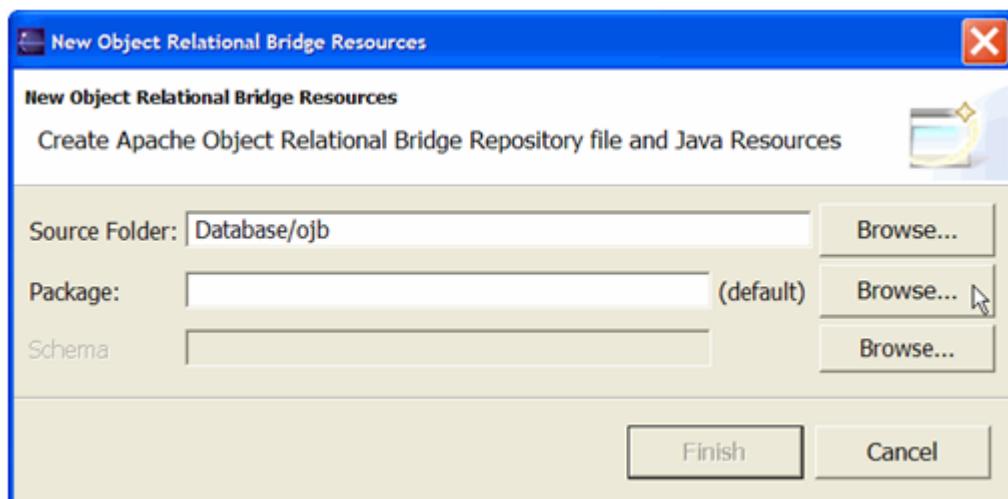
The wizard will be opened with a selected Database Connection if applicable.



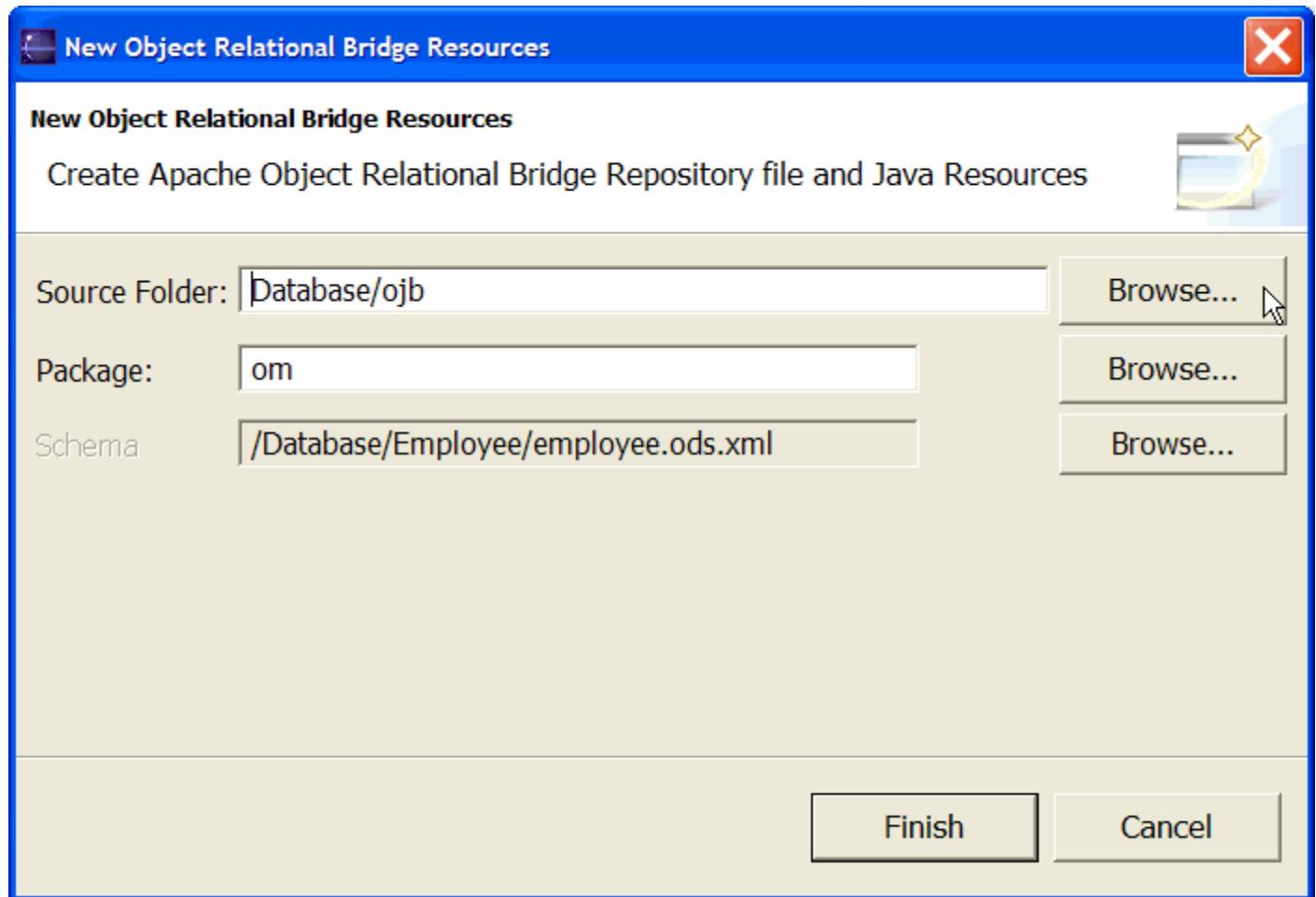
Otherwise:



The wizard will be opened without any selected Database connection



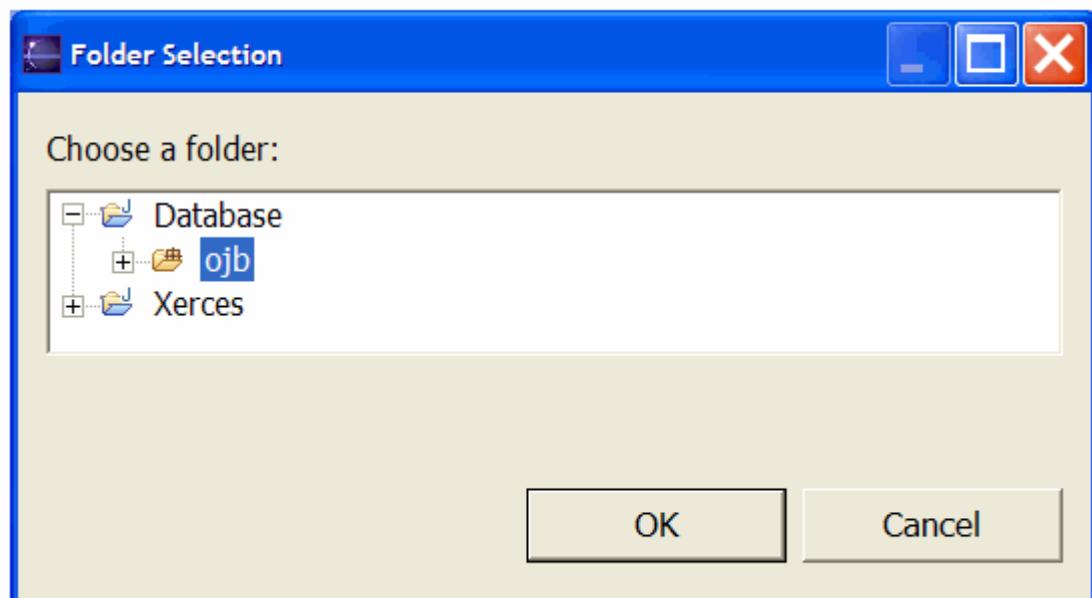
2.1. Source Folder



Type in the Java Source Folder text field the targeted Java Source Folder.

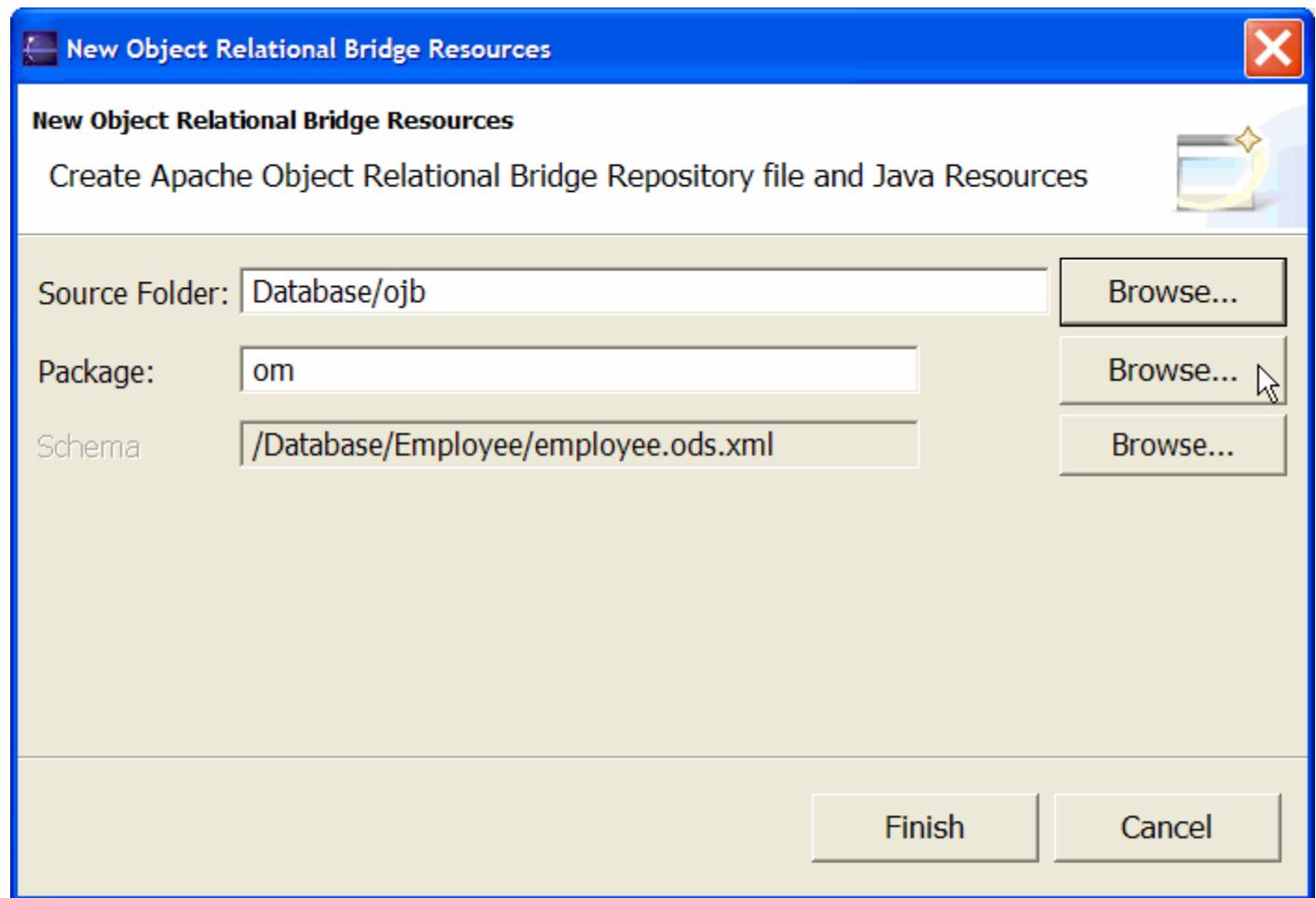
The Java Source Folder should exist.

Otherwise use the **Browse** button :

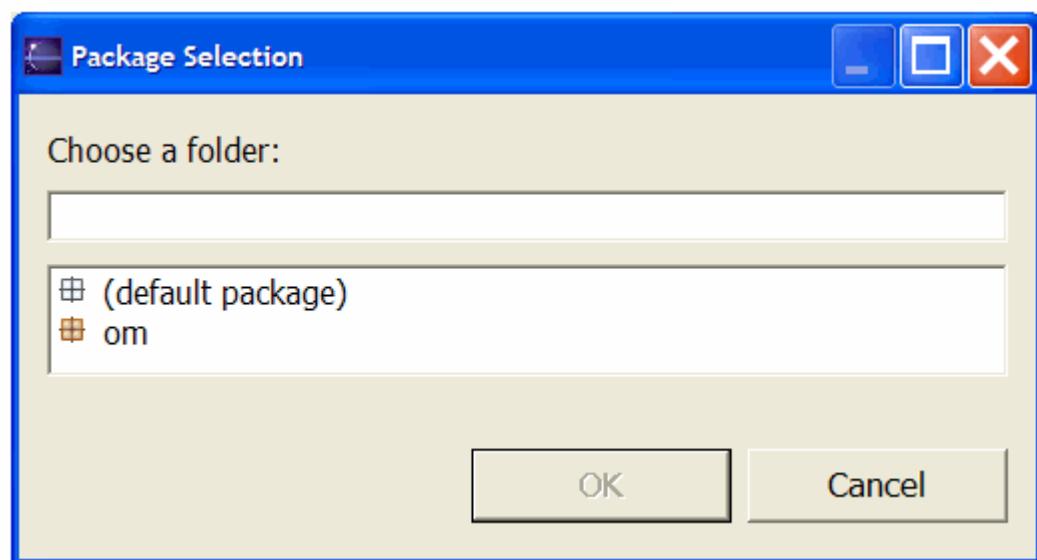


Select the appropriate Java Source Folder.

2.2. Package

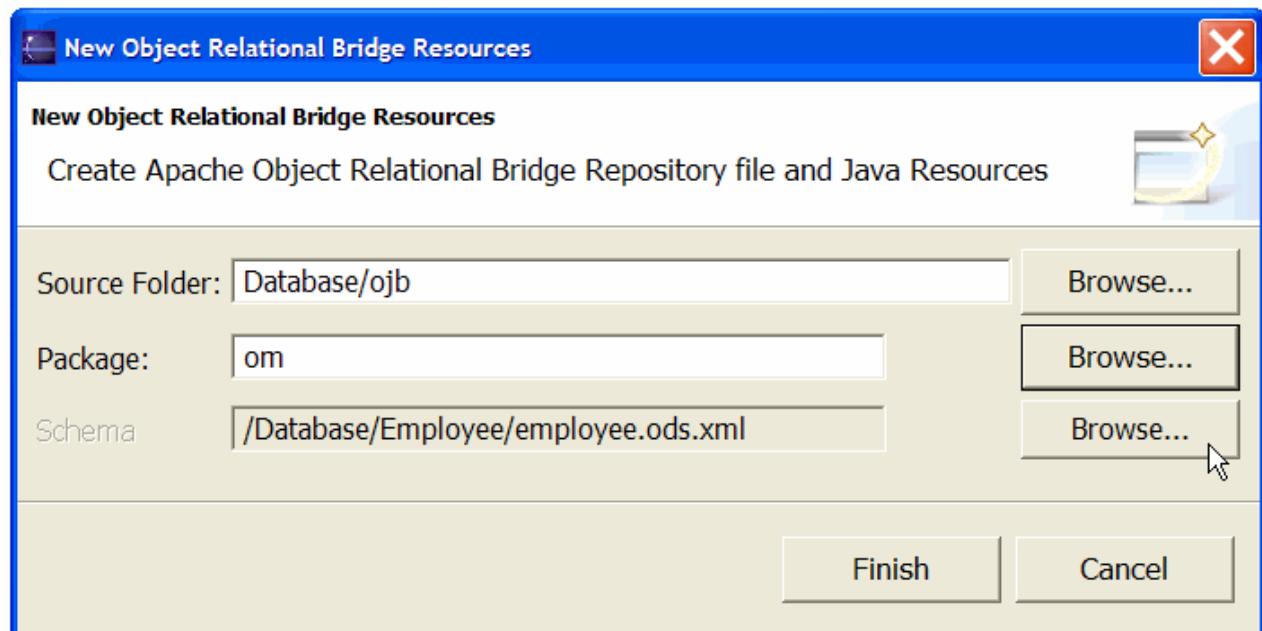


Type in the Package text field the targeted Java Package.
Otherwise use the **Browse** button :

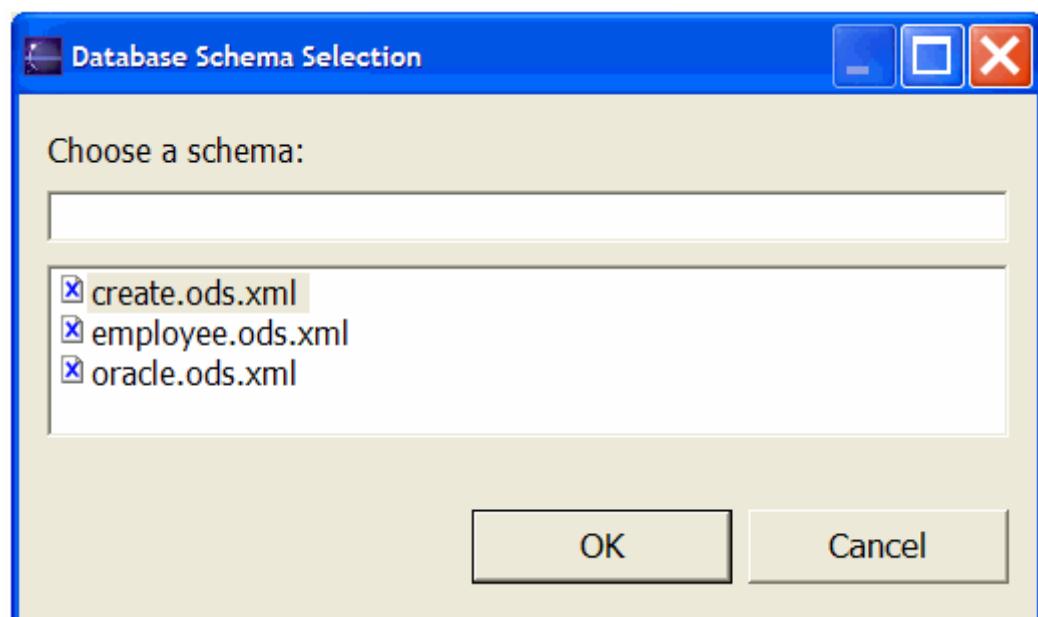


Select the appropriate targeted Java Package.

2.3. Schema



Use the **Browse** button :

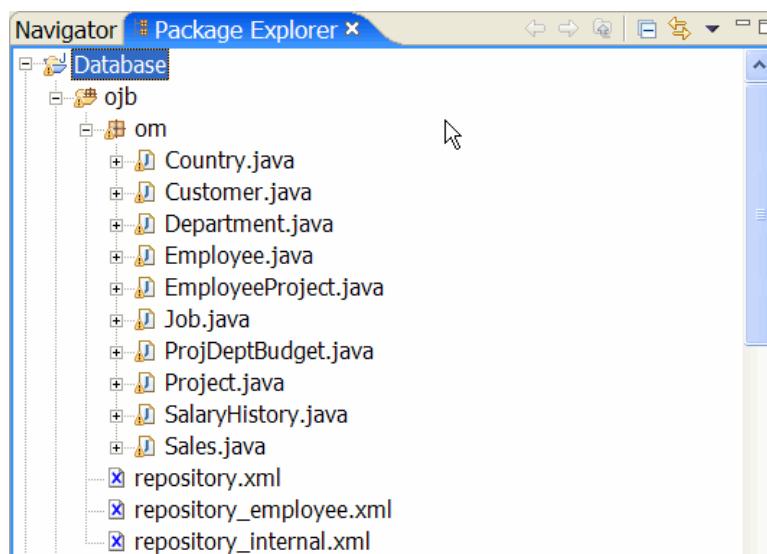


The Database Schema selector displays the available Database Schema in your current Project.

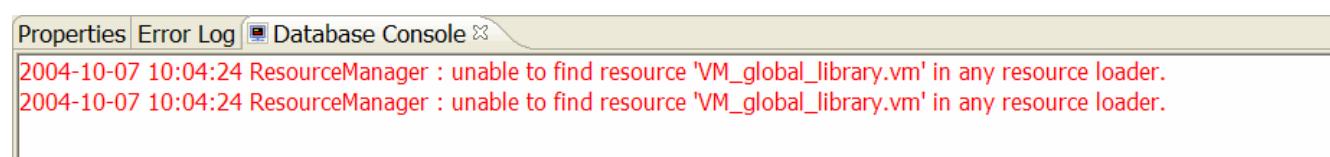
Select the appropriate Database Schema.

2.4. Finish

The code generation process will be performed in the targeted Java Source Folder and its designated target package



Detailed information will be displayed in the [DatabaseConsole](#) window.

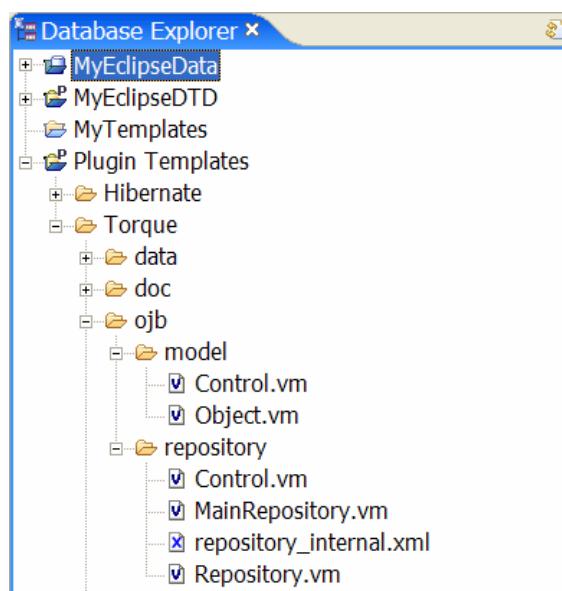


3. Templates

This wizard is [Template](#) based.

Below your [Plugin Templates](#) or [MyTemplates](#) you should see the following directory.

Torque/obj



These templates are [Velocity](#) based and use the [Torque Model API](#).

Hibernate

1. [Introduction](#)
2. [Hibernate Configuration and Mapping Resource](#)
3. [Hibernate Java Resources](#)

1. Introduction

You will learn in this chapter how to generate Hibernate resources.

EclipseDatabase proposes two wizards who will help you to generate your Hibernate resources.

- [Hibernate Configuration and Mapping Resources](#)
- [Hibernate Java Resources](#)

Generalities about the way EclipseDatabase manage the wizards have been explained in the [Modeling in the Workspace Introduction](#) chapter.

Hibernate Configuration and Mapping Ressources

1. [Introduction](#)
2. [Hibernate Configuration and Mapping Resources](#)
 1. [Source Folder](#)
 2. [Package](#)
 3. [Schema](#)
 4. [Options](#)
 1. [Multiple Files](#)
 2. [XDoclet Support](#)
 5. [Finish](#)
3. [Templates](#)
4. [Additional Resources](#)

1. Introduction

The purpose of this chapter is to show how to generate Hibernate Configuration and Mapping Resources.

This wizard will generate :

- [Hibernate Configuration file](#)
- [Hibernate Mapping files](#)

This wizard needs to use a target Java Project.

3. Hibernate Configuration and Mapping Resources

To start the Hibernate Configuration and Mapping Resources wizard, select :

File->New->Other->Database->Hibernate Configuration and Mapping Resources

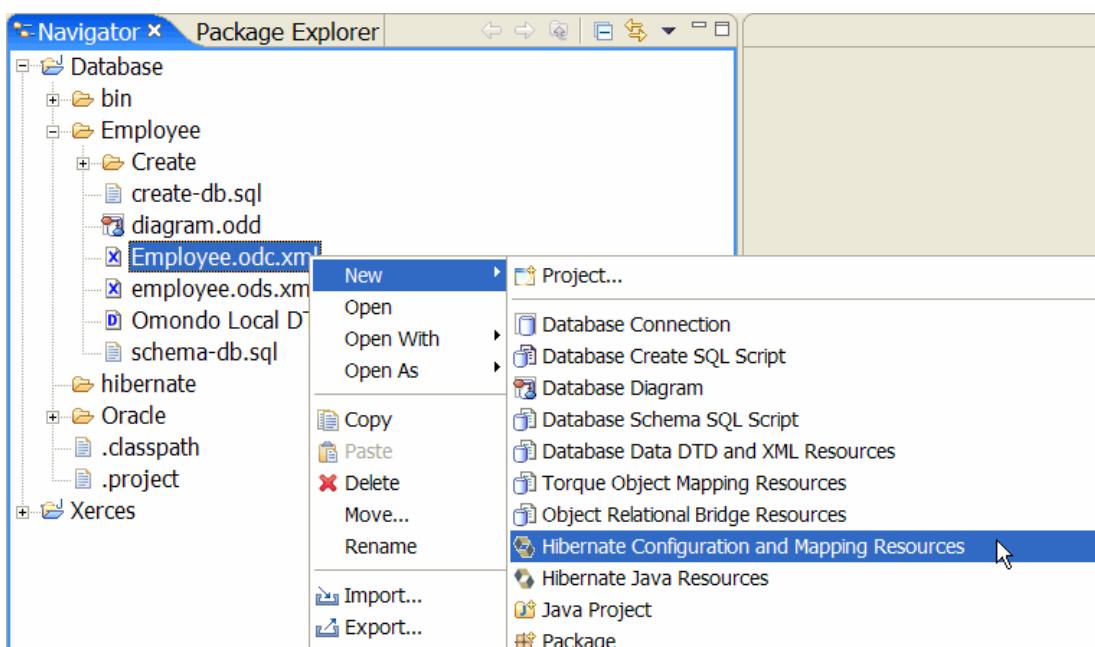
or

Select an object in the Package Explorer View or the Navigator View then right-click and select :

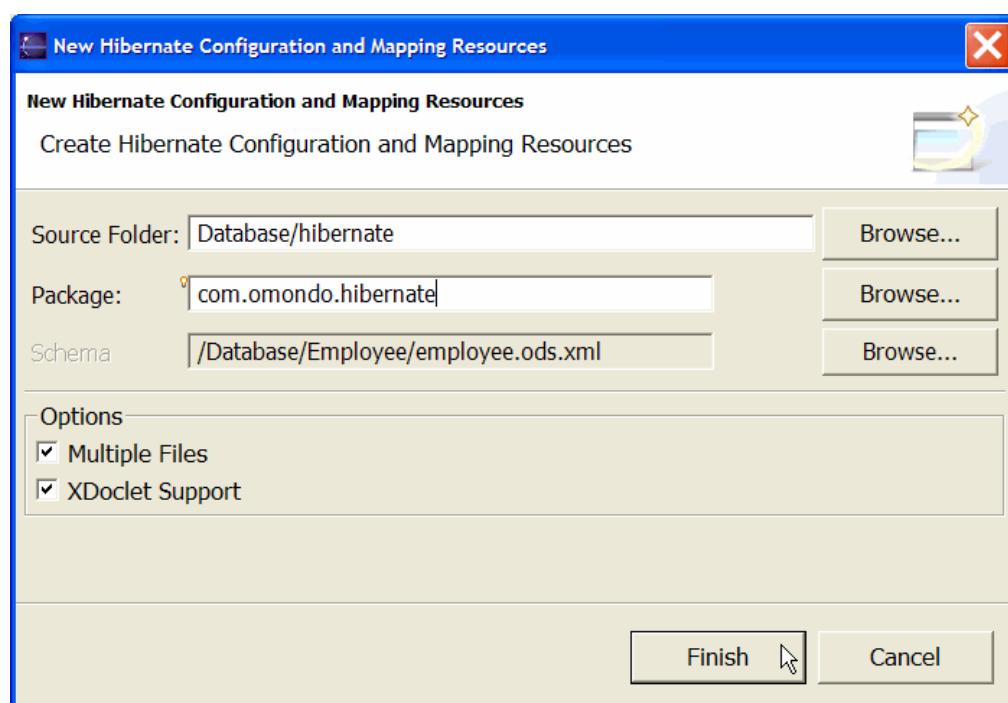
New->Other->Database->Hibernate Configuration and Mapping Resources

The selection is contextual.

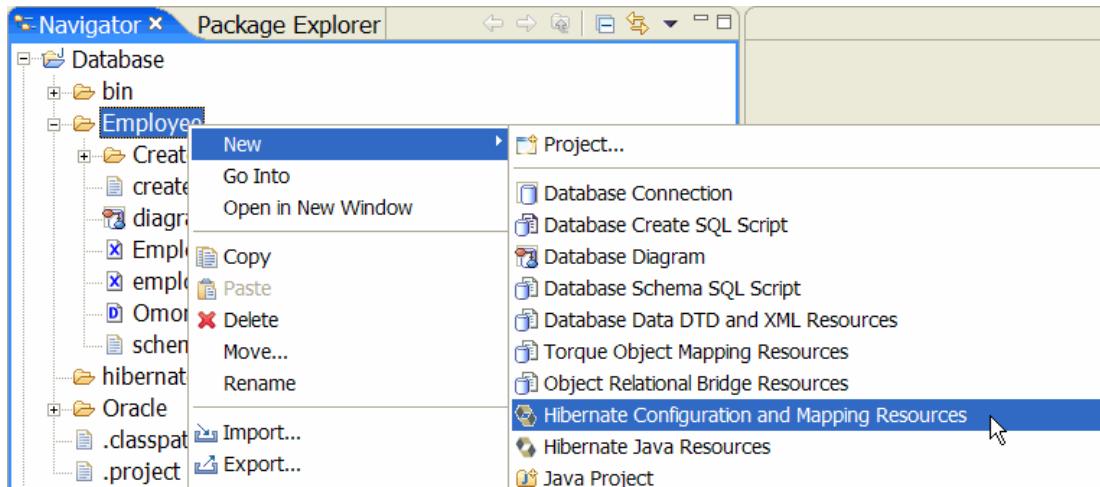
This means that if you select an existing Database Object:



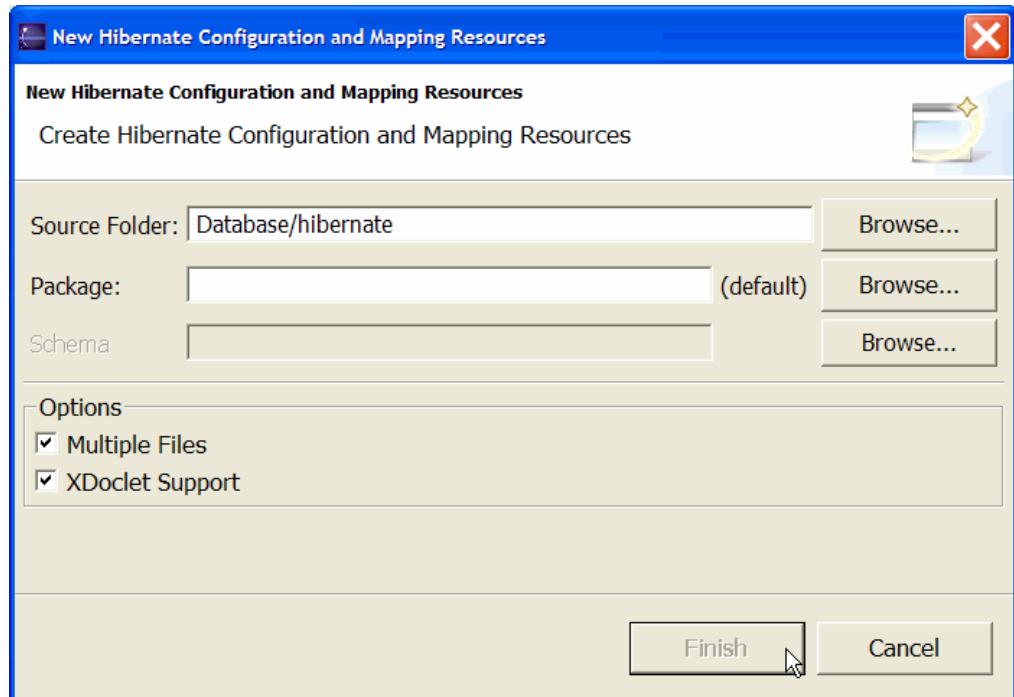
The wizard will be opened with a selected Database Connection if applicable.



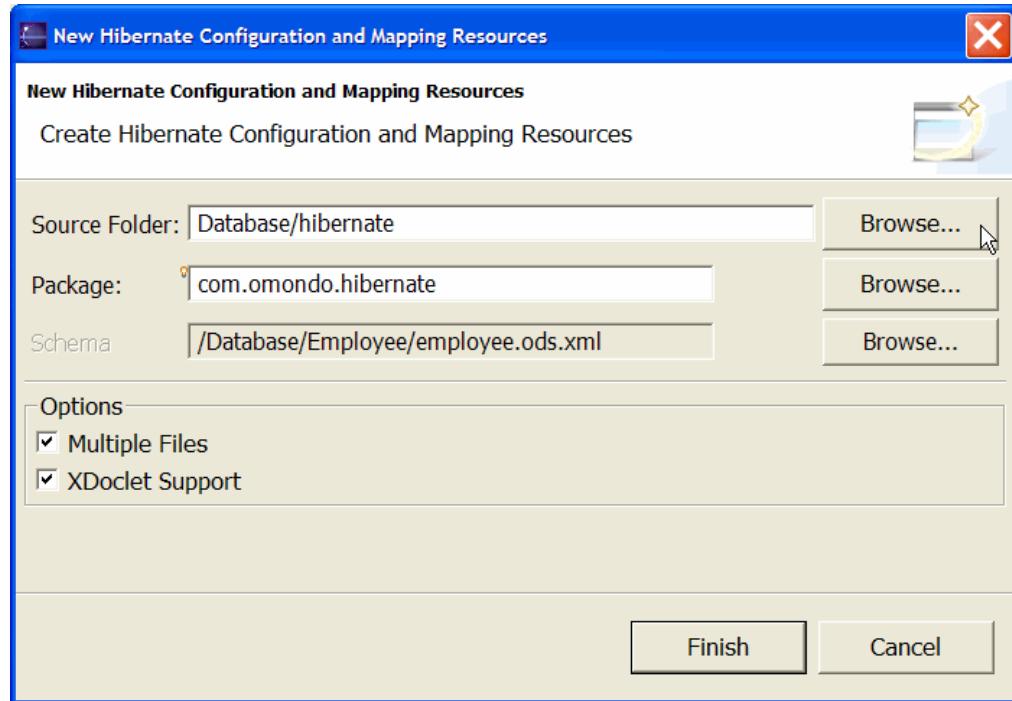
Otherwise:



The wizard will be opened without any selected Database Connection.



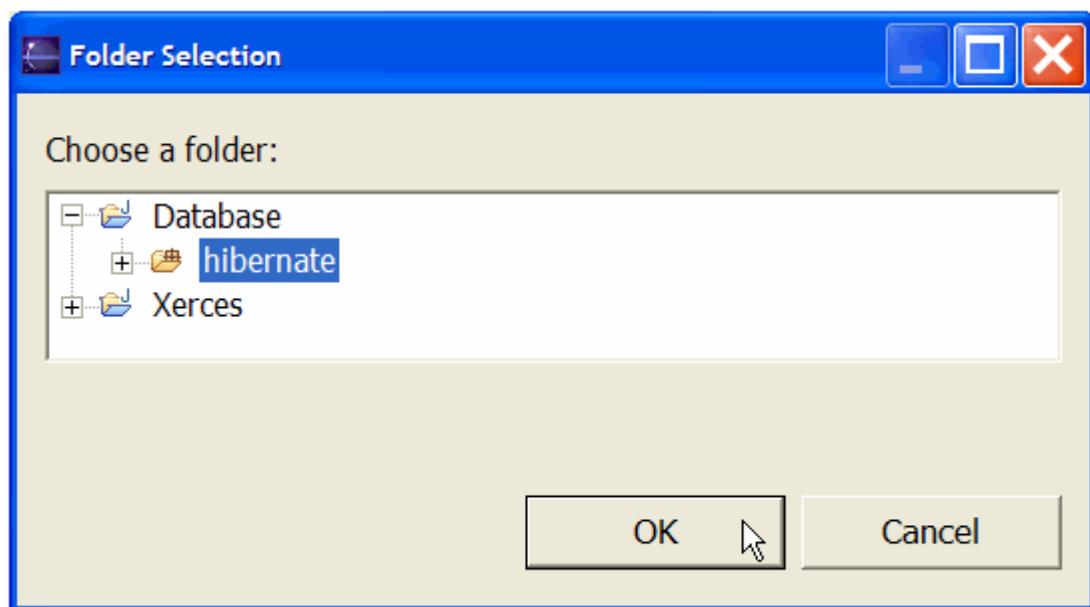
2.1. Source Folder



Type in the Java Source Folder text field the targeted Java Source Folder.

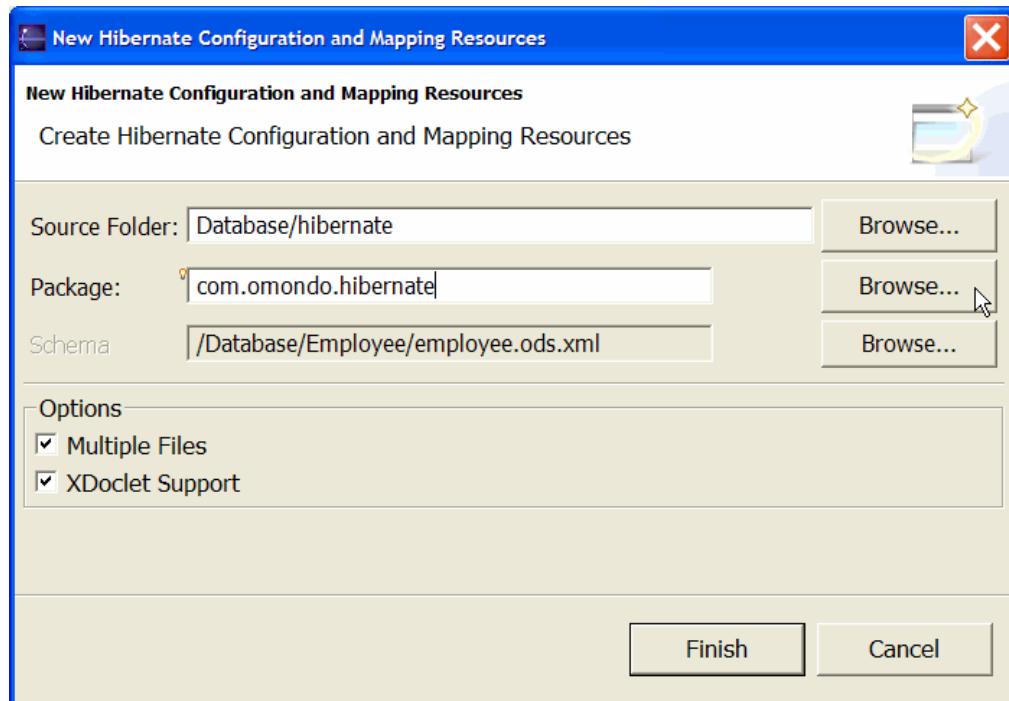
The Java Source Folder should exist.

Otherwise use the **Browse** button :



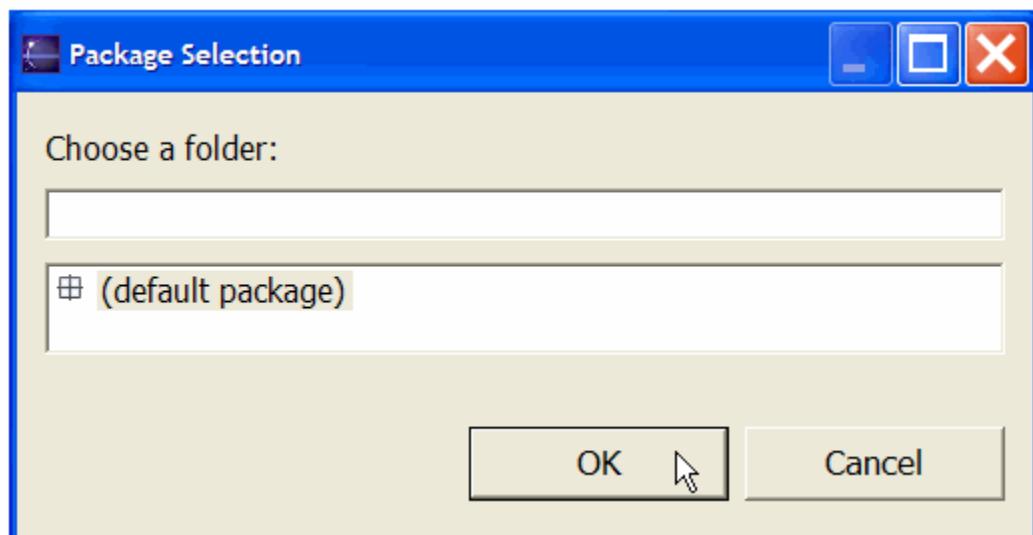
Select the appropriate Java Source Folder.

2.2. Package



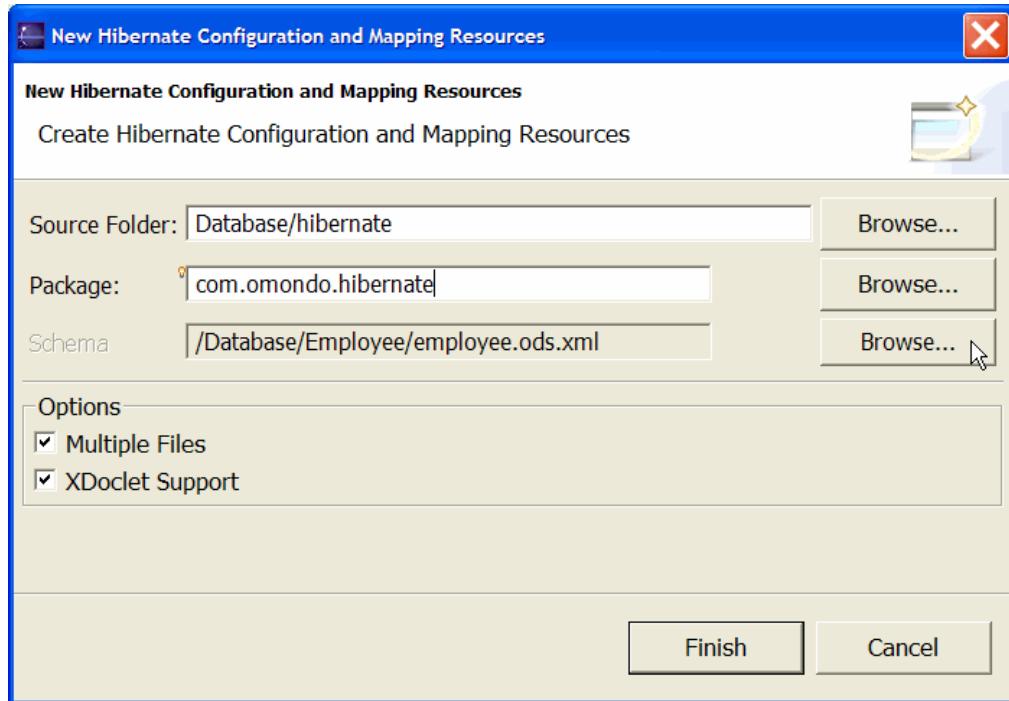
Type in the Package text field the targeted Java Package.

Otherwise use the **Browse** button :

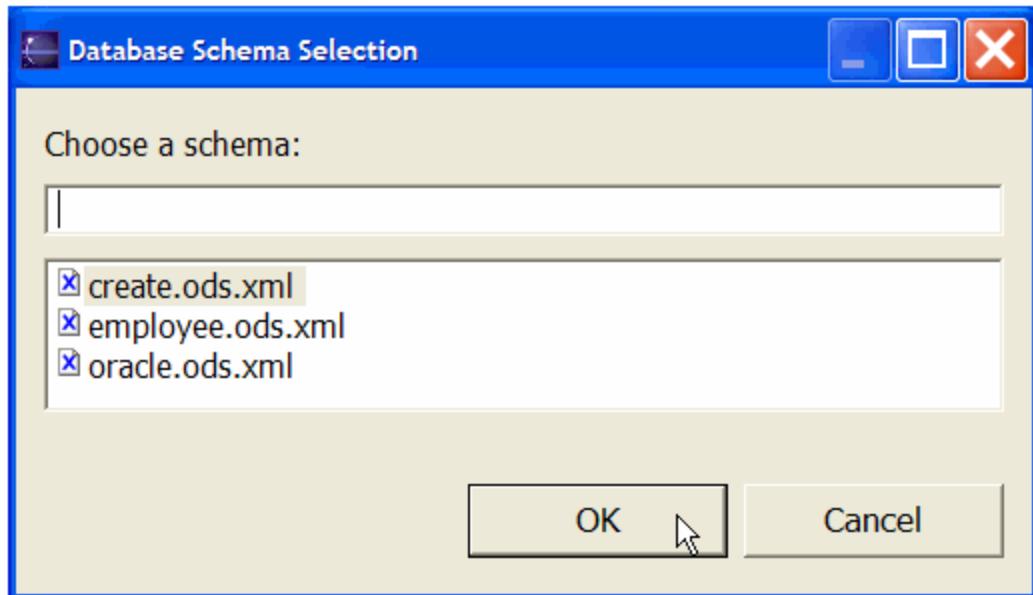


Select the appropriate targeted Java Package.

2.3. Schema



Use the **Browse** button :



The Database Schema selector displays the available Database Schema in your current Project.

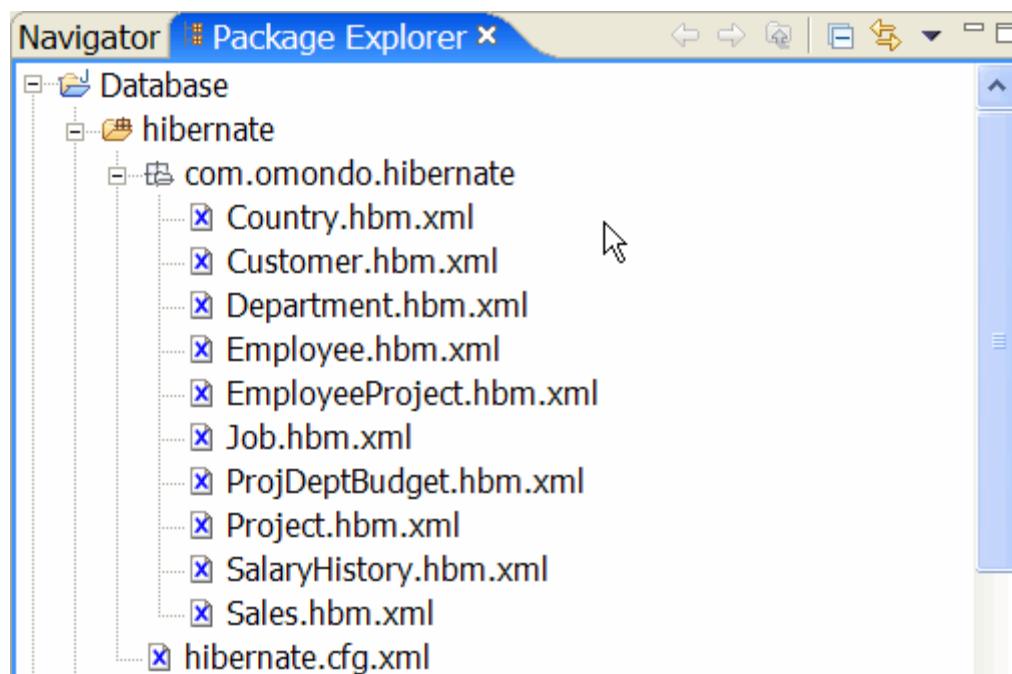
Select the appropriate Database Schema.

2.4. Options

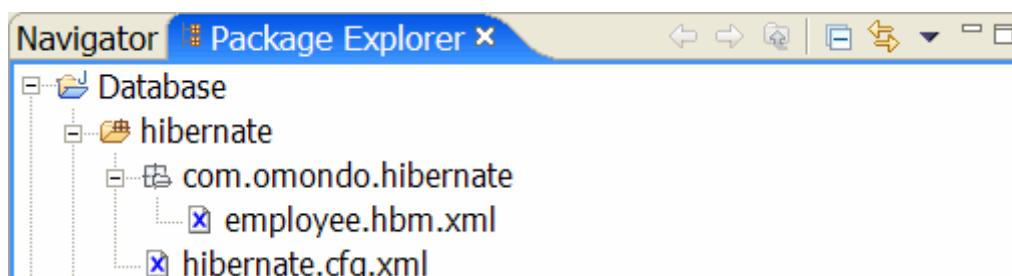
You can set various Hibernate options.

2.4.1. Multiple Files

Each Database Table will see its Hibernate Mapping definition in its own file.

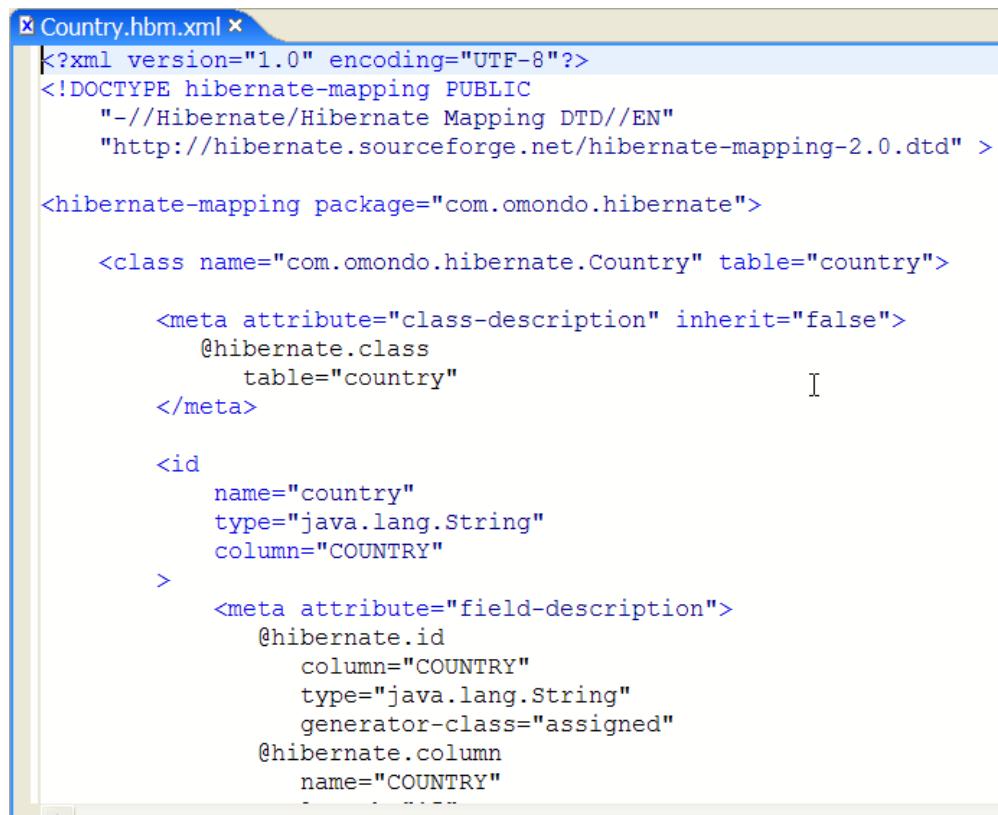


Otherwise only one Hibernate Mapping file will be generated.



2.4.2. XDoclet Support

If checked, Hibernate Mapping file will contain XDoclet attributes.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd" >

<hibernate-mapping package="com.omondo.hibernate">

  <class name="com.omondo.hibernate.Country" table="country">

    <meta attribute="class-description" inherit="false">
      @hibernate.class
      table="country"
    </meta>

    <id
      name="country"
      type="java.lang.String"
      column="COUNTRY"
    >
      <meta attribute="field-description">
        @hibernate.id
        column="COUNTRY"
        type="java.lang.String"
        generator-class="assigned"
        @hibernate.column
        name="COUNTRY"
      </meta>
    </id>
  </class>
</hibernate-mapping>
```

Otherwise, no XDoclet attributes will be generated.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd" >

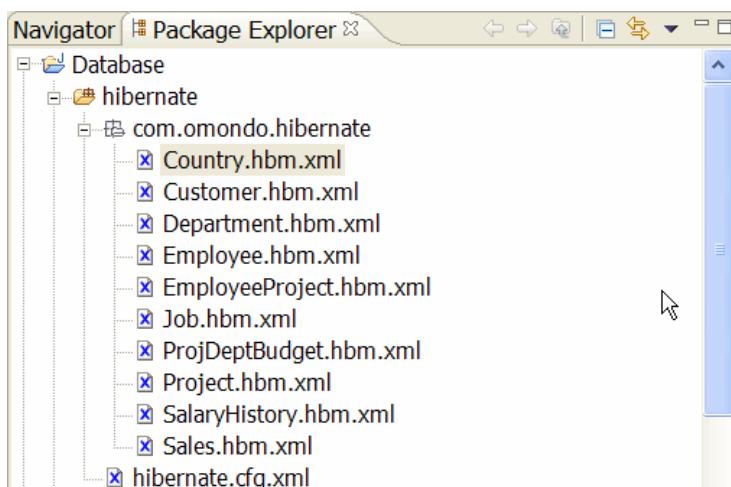
<hibernate-mapping package="com.omondo.hibernate">

  <class name="com.omondo.hibernate.Country" table="country">

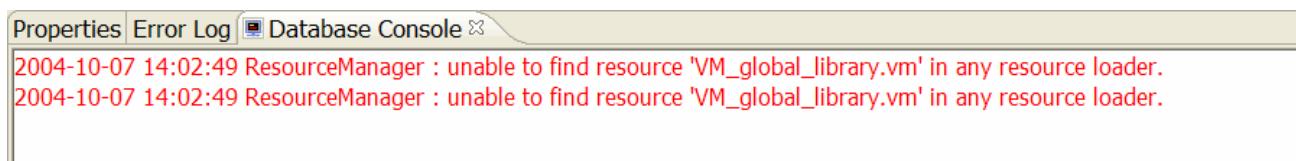
    <id
      name="country"
      type="java.lang.String"
      column="COUNTRY"
    >
      <column
        name="COUNTRY"
        length="15"
        not-null="true"
        unique="true"
        sql-type="VARCHAR"
      />
      <generator class="assigned"/>
    </id>
  </class>
</hibernate-mapping>
```

2.5. Finish

The Hibernate Configuration and Mapping generation process will be performed in the targeted Java Source Folder and its designated target package.



Detailed information will be displayed in the [DatabaseConsole](#) window.



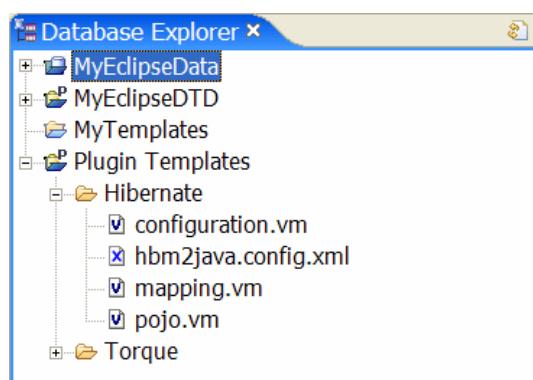
3. Templates

This wizard is [Template](#) based.

Below your [Plugin Templates](#) or [MyTemplates](#) you should see the following Velocity files:

Torque/Hibernate/configuration.vm

Torque/Hibernate/mapping.vm



These templates are [Velocity](#) based and use the EclipseDatabase API.
This API is not yet published.

Hibernate Java Resources

1. Introduction
2. Hibernate Java Resources
 1. Source Folder
 2. Configuration
 3. Options
 1. Generate Hibernate Code
 1. Use Basic Renderer
 1. Generate Empty Concrete Classes
 2. Generate Basic Finder
 2. Use Velocity Renderer
 2. Generate Stateless Session Bean
 3. Use hbm2java Config
 1. Use Internal Config
 2. Configuration
 4. Finish
 3. Additional Resources

1. Introduction

The purpose of this chapter is to show how to generate Hibernate Java classes.
This wizard needs to use a target Java Project.

This wizard is an integration of the separate [Hibernate Extensions hbm2java](#).
EclipseDatabase provides the current embedded [Hibernate Extensions API](#).

2. Hibernate Java Resources

To start the Hibernate Configuration and Mapping Resources wizard, select :

File->New->Other->Database->Hibernate Java Resources

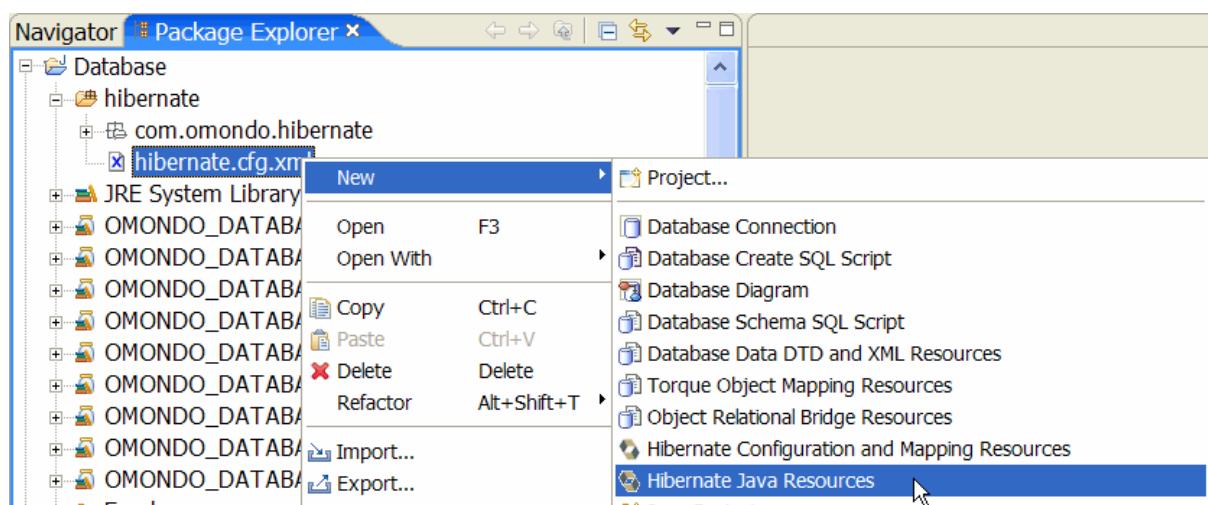
or

Select an object in the Package Explorer View or the Navigator View then right-click and select :

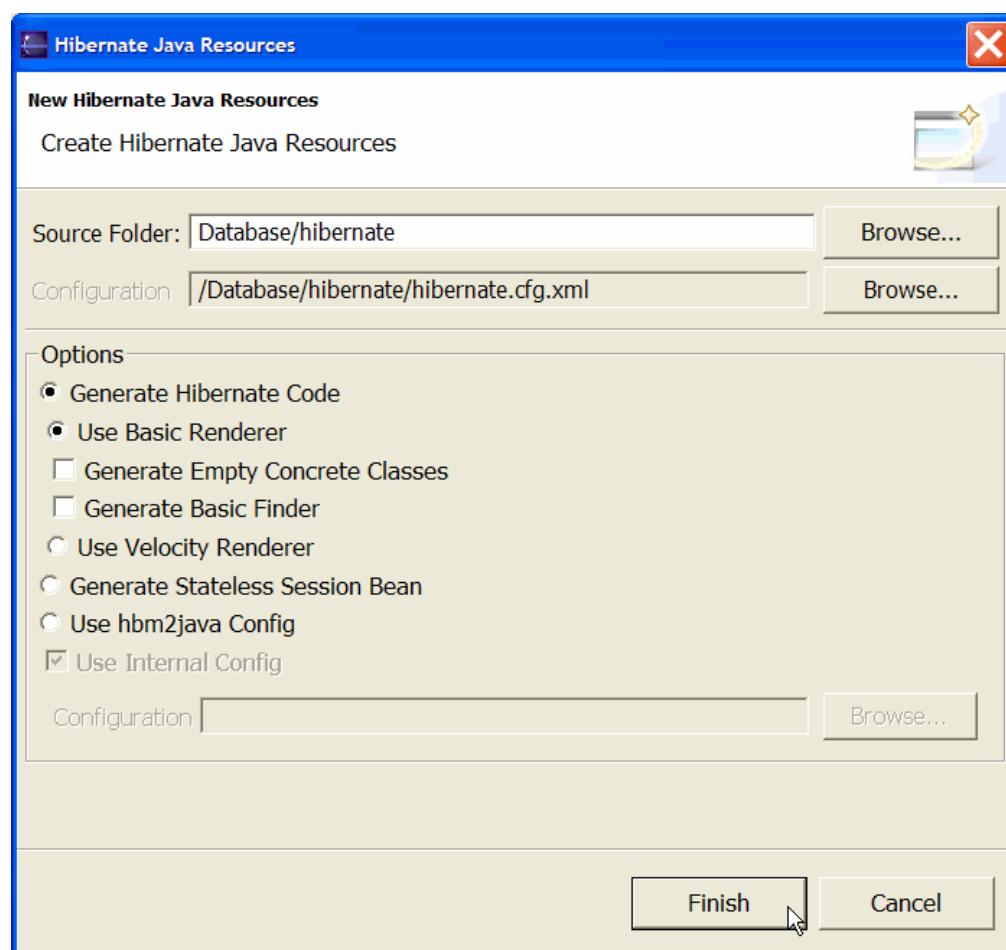
New->Other->Database->Hibernate Java Resources

The selection is contextual.

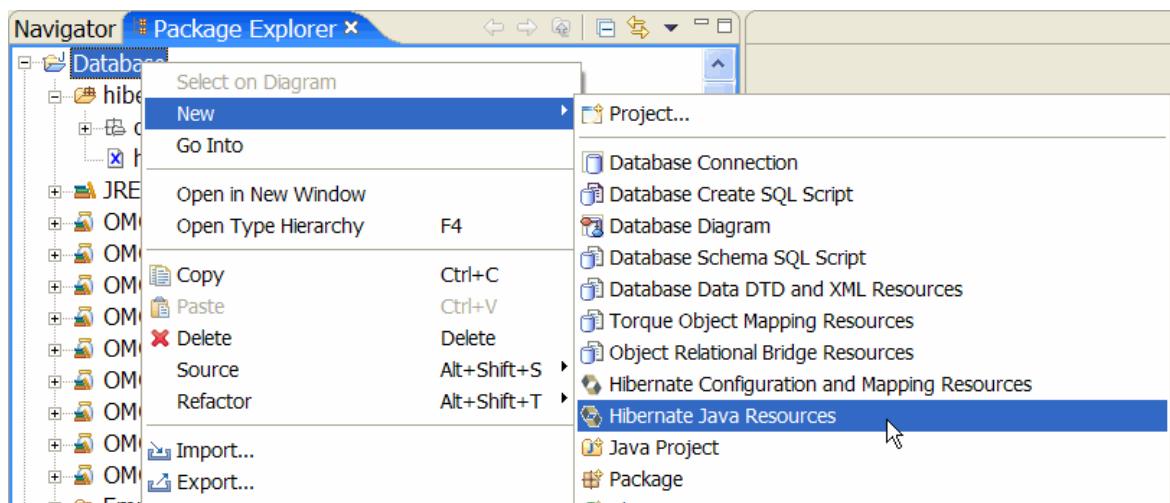
This means that if you select an existing Hibernate Configuration file.



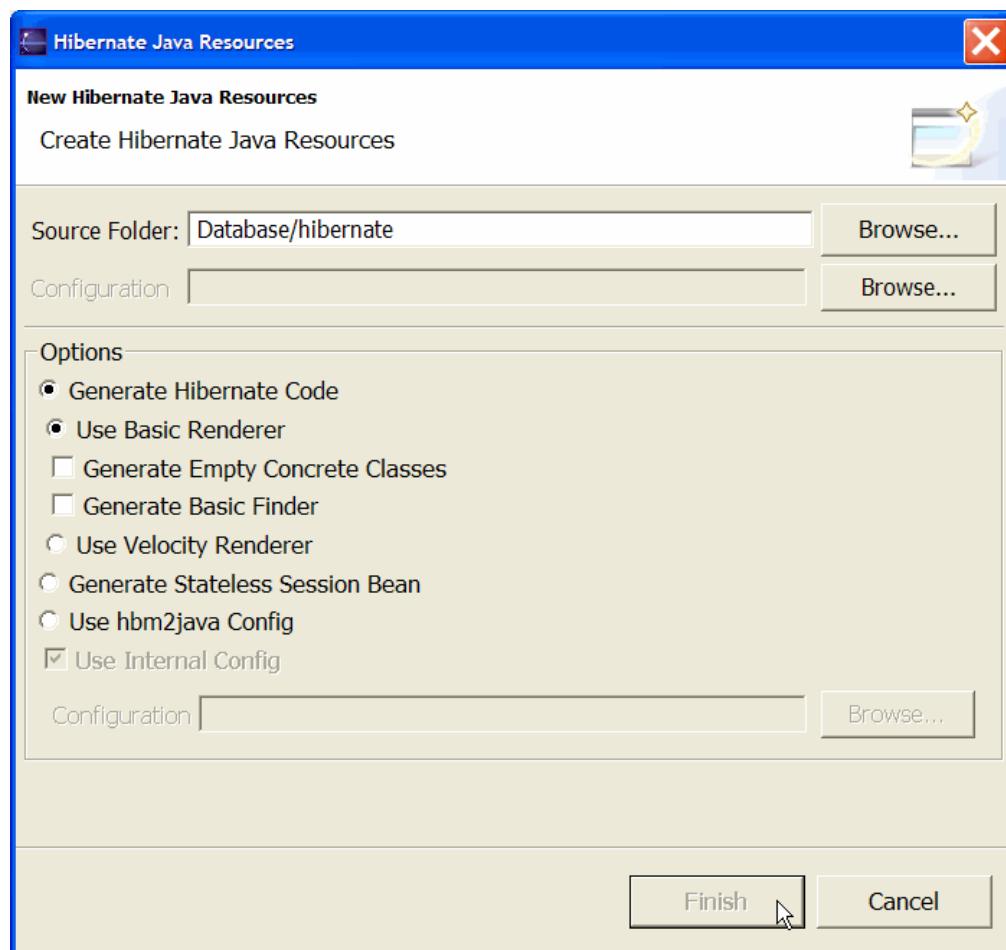
The wizard will be opened with a selected Hibernate Configuration file if applicable.



Otherwise :



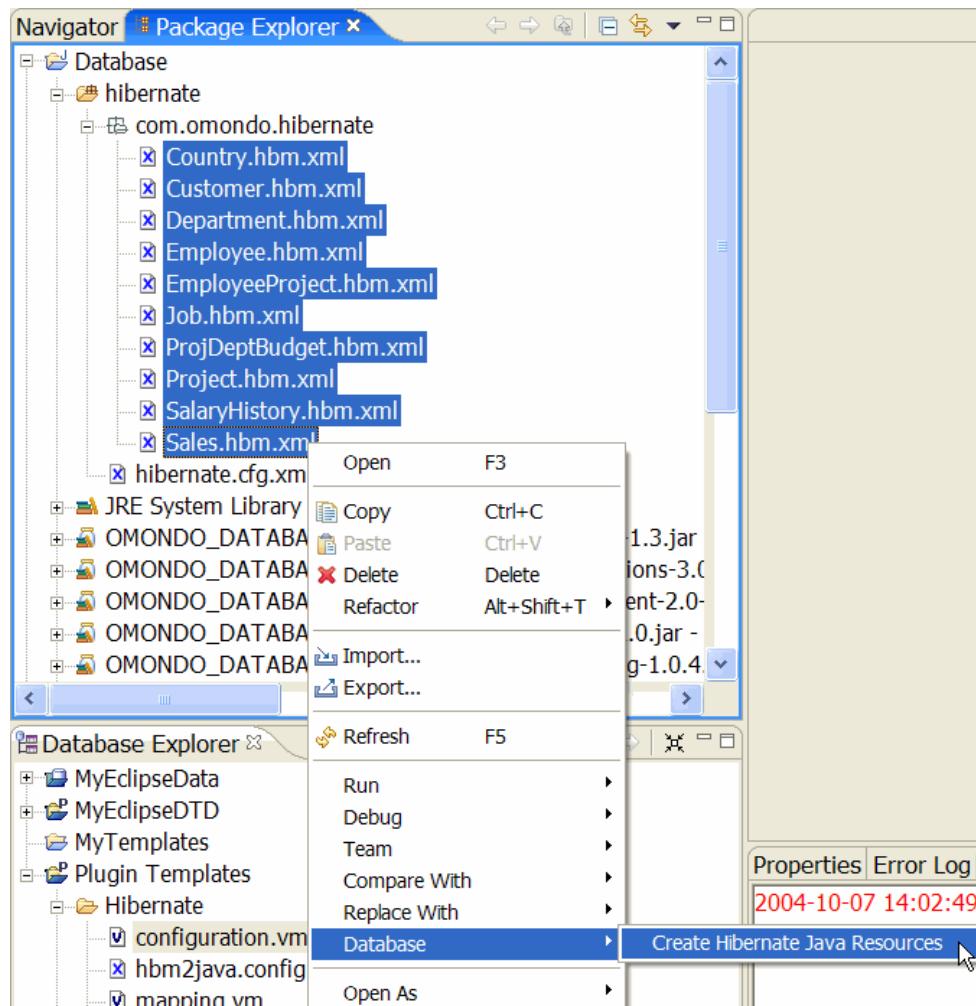
The wizard will be opened without any selected Hibernate Configuration file.



You can run this wizard in another way.

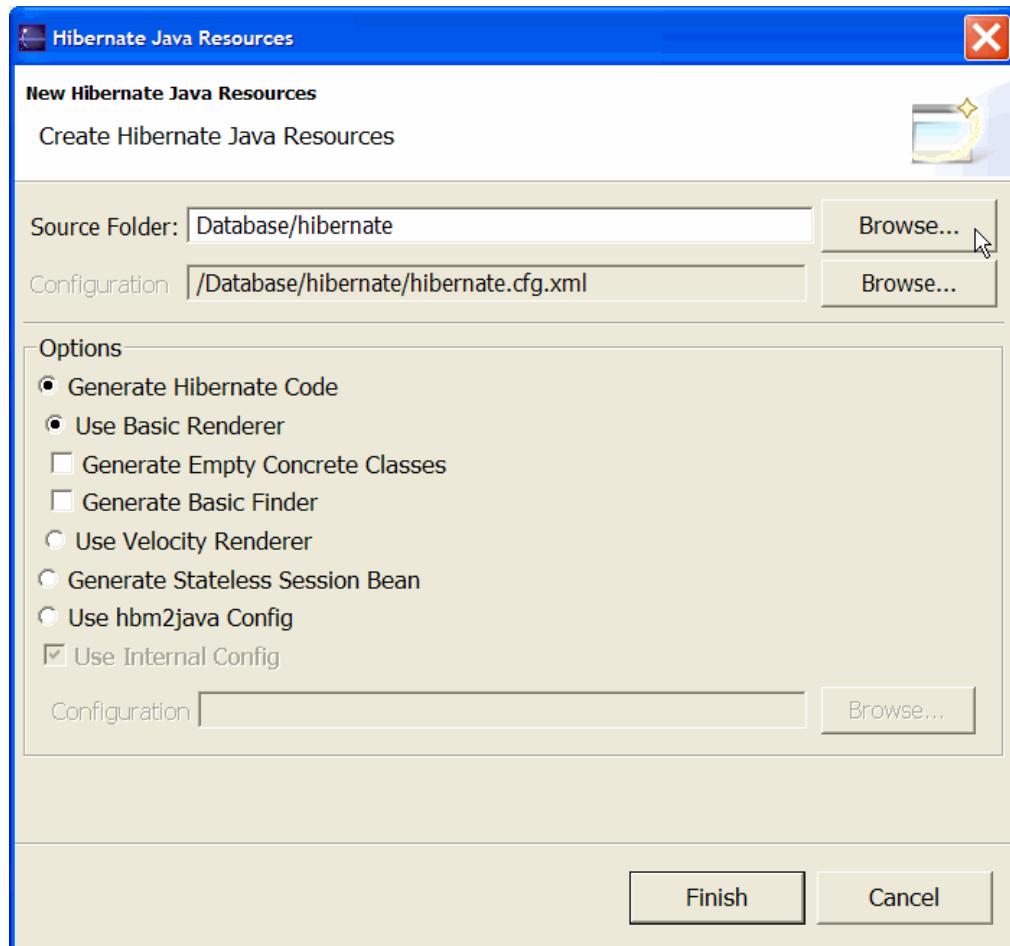
Select a set of Hibernate Mapping file definitions, right-click and select :

Create Hibernate Java Resources



However the Configuration selector will not be available.

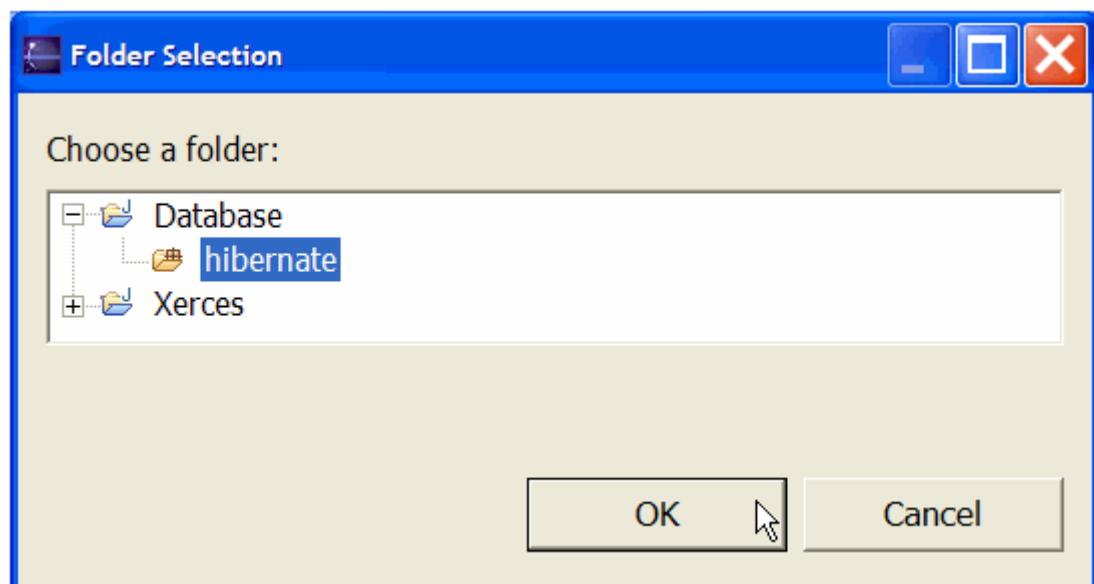
2.1. Source Folder



Type in the Java Source Folder text field the targeted Java Source Folder.

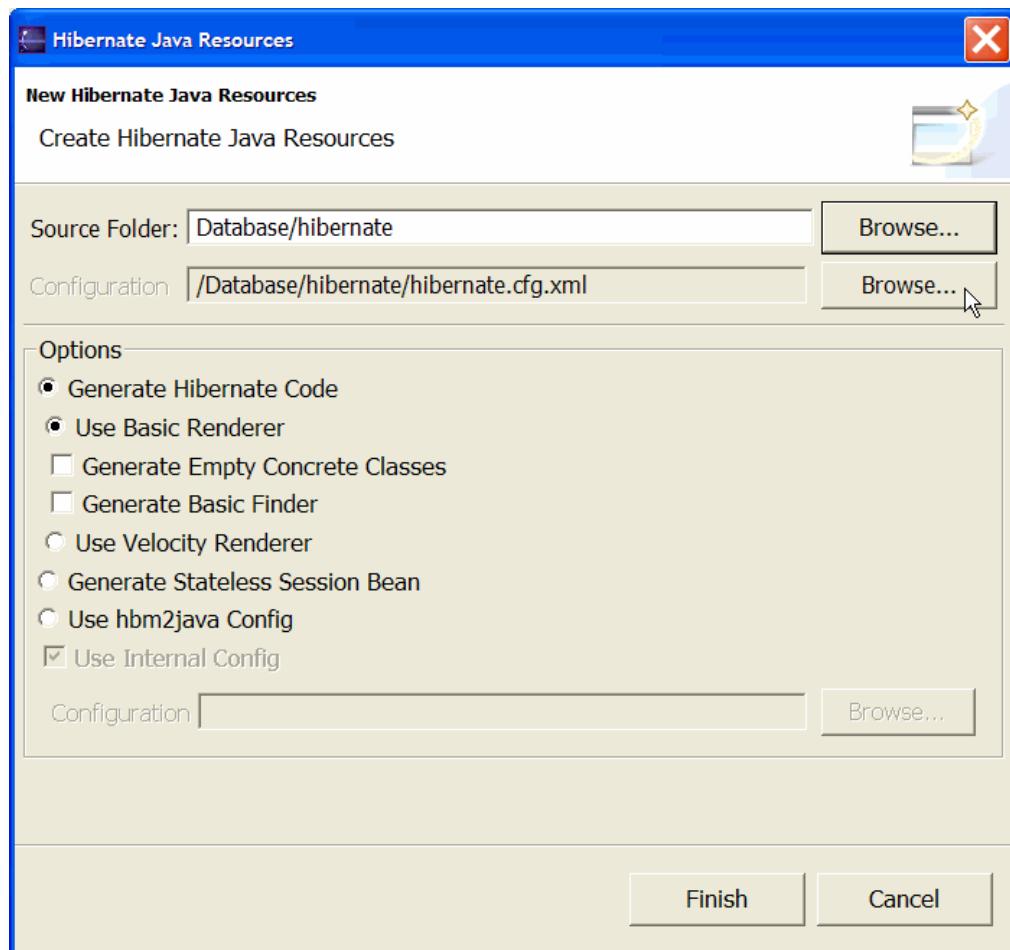
The Java Source Folder should exist.

Otherwise use the **Browse** button :

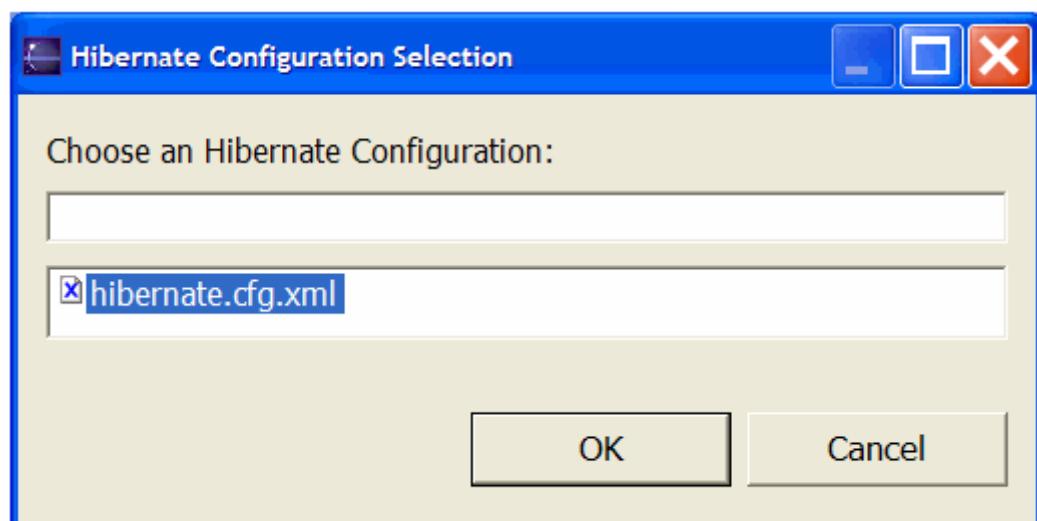


Select the appropriate Java Source Folder.

2.2. Configuration



Use the **Browse** button :



Select the appropriate Hibernate Configuration file.

2.3. Options

You can set various Hibernate options.

2.3.1. Generate Hibernate Code

The **hbm2java** Hibernate code generator works in two ways :

The first one is an internal code generator, [Basic Renderer](#).

The second one uses a [Velocity](#) templates, [Velocity Renderer](#).

2.3.1.1. Use Basic Renderer

This code generator is the preferred Hibernate code generator.

2.3.1.1.1. Generate Empty Concrete Classes

With this option, the wizard will generate two sets of classes.

One set of Abstract classes.

One set of Concrete classes.

With this option you will obtain a clean separation between your code and the generated code.

2.3.1.1.2. Generate Basic Finder

With this option, the wizard will generate three sets of classes.

One set of Abstract classes.

One set of Concrete classes.

One set of Finder classes.

With this option you will obtain a clean separation between your code and the generated code.

2.3.1.2. Use Velocity Renderer

Below your [Plugin Templates](#) or [MyTemplates](#) you should see the following file.

Hibernate/pojo.vm

If checked, the wizard will use this velocity template to generate Hibernate Java classes.

2.3.2. Generate Stateless Session Bean

The hbm2java tool contains a Stateless Session Bean generator.

This generator uses as its input a set of Hibernate files.

2.3.3. Use hbm2java Config

You can use an optional xml file to configure this wizard.

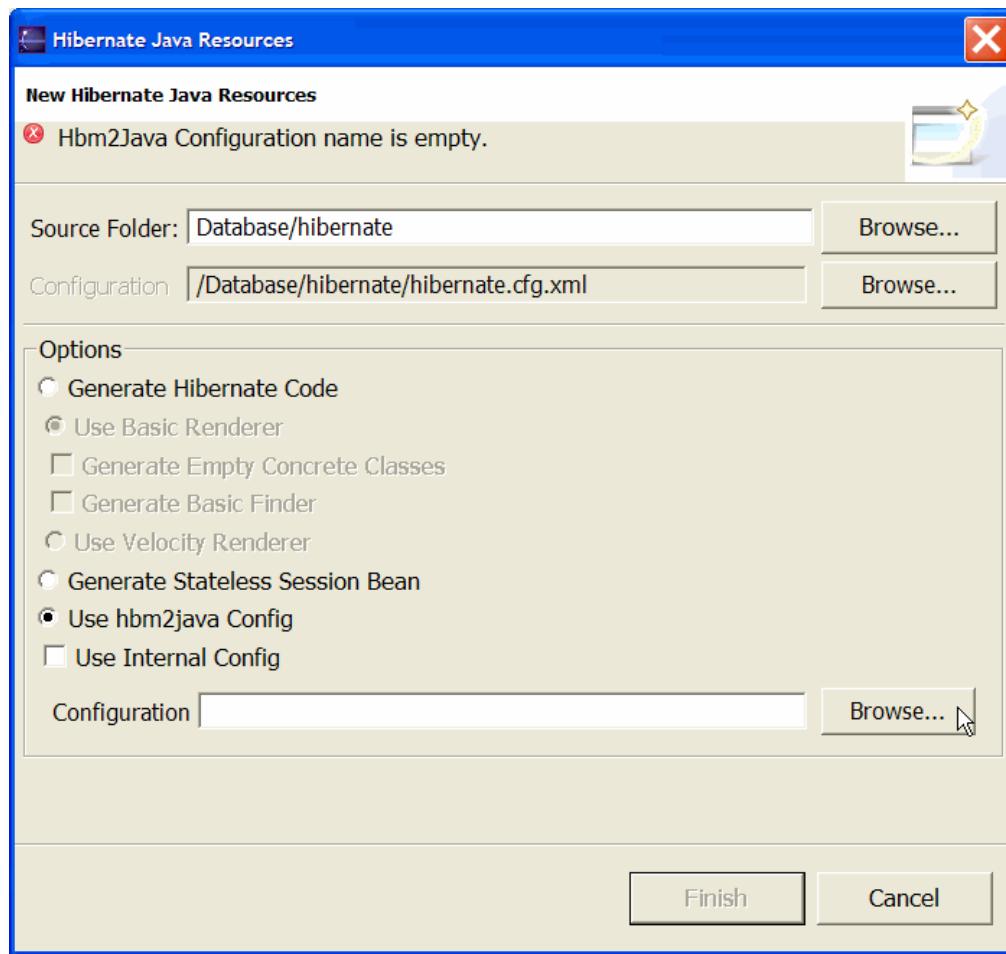
2.3.3.1. Use Internal Config

Below your [Plugin Templates](#) or [MyTemplates](#) you should see the following file:

Hibernate/hbm2java.config.xml

If checked, the wizard will use this **hbm2java.config.xml** file.

2.3.3.2. Configuration

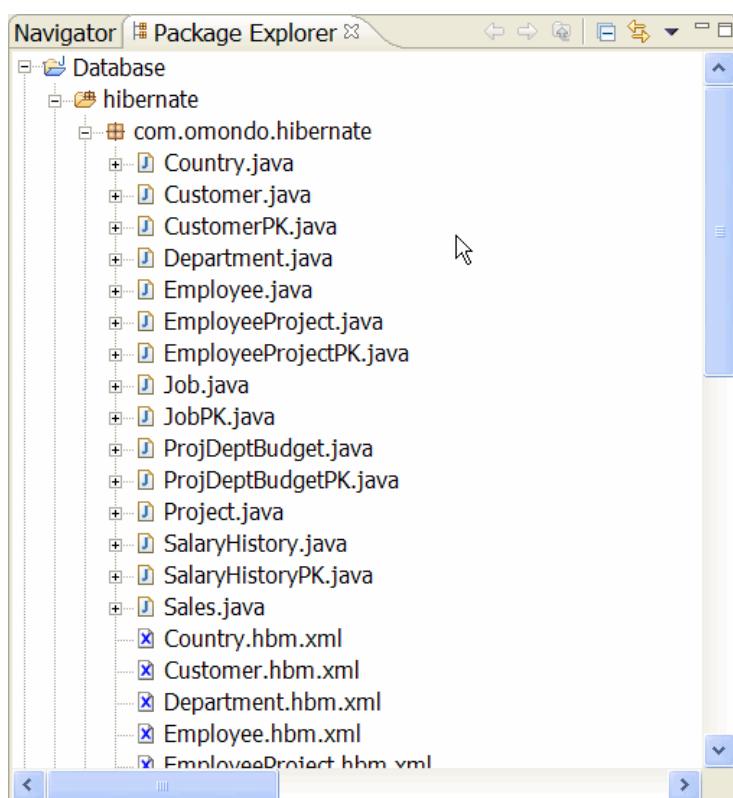


The **Browse** button lets you locate an **.xml** file.

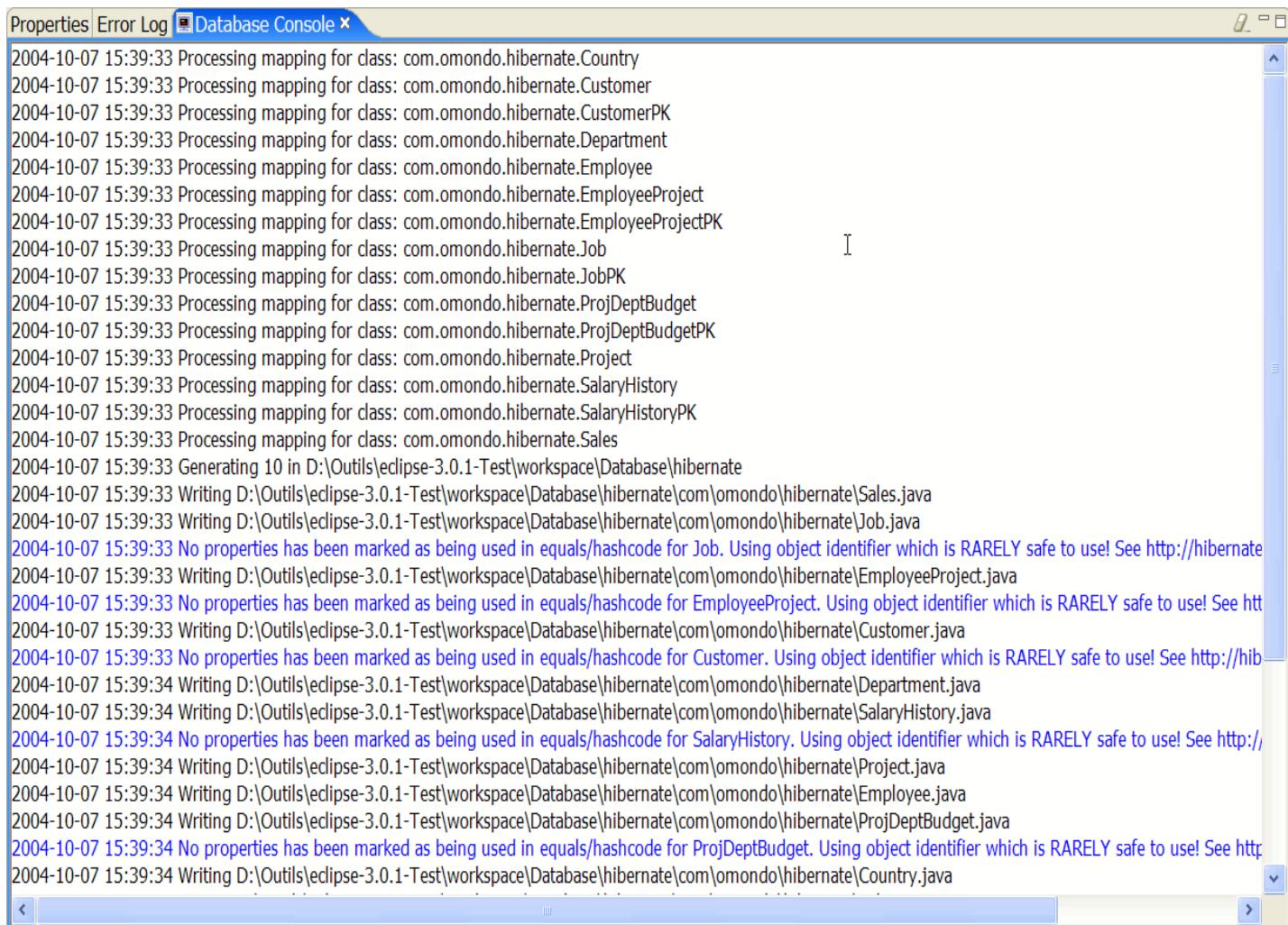
Usually you locate **hbm2java.config.xml** file.

2.4. Finish

The code generation process will be performed in the targeted Java Source Folder



Detailed information will be displayed in the **DatabaseConsole** window.



The screenshot shows the Eclipse Database Console window with the title bar "Properties Error Log Database Console". The main area displays a log of processing mappings for various Hibernate classes. The log entries are as follows:

```

2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.Country
2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.Customer
2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.CustomerPK
2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.Department
2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.Employee
2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.EmployeeProject
2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.EmployeeProjectPK
2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.Job
2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.JobPK
2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.ProjDeptBudget
2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.ProjDeptBudgetPK
2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.Project
2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.SalaryHistory
2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.SalaryHistoryPK
2004-10-07 15:39:33 Processing mapping for class: com.omondo.hibernate.Sales
2004-10-07 15:39:33 Generating 10 in D:\Outils\eclipse-3.0.1-Test\workspace\Database\hibernate
2004-10-07 15:39:33 Writing D:\Outils\eclipse-3.0.1-Test\workspace\Database\hibernate\com\omondo\hibernate\Sales.java
2004-10-07 15:39:33 Writing D:\Outils\eclipse-3.0.1-Test\workspace\Database\hibernate\com\omondo\hibernate\Job.java
2004-10-07 15:39:33 No properties has been marked as being used in equals/hashcode for Job. Using object identifier which is RARELY safe to use! See http://hibernate
2004-10-07 15:39:33 Writing D:\Outils\eclipse-3.0.1-Test\workspace\Database\hibernate\com\omondo\hibernate\EmployeeProject.java
2004-10-07 15:39:33 No properties has been marked as being used in equals/hashcode for EmployeeProject. Using object identifier which is RARELY safe to use! See ht
2004-10-07 15:39:33 Writing D:\Outils\eclipse-3.0.1-Test\workspace\Database\hibernate\com\omondo\hibernate\Customer.java
2004-10-07 15:39:33 No properties has been marked as being used in equals/hashcode for Customer. Using object identifier which is RARELY safe to use! See http://hib
2004-10-07 15:39:34 Writing D:\Outils\eclipse-3.0.1-Test\workspace\Database\hibernate\com\omondo\hibernate\Department.java
2004-10-07 15:39:34 Writing D:\Outils\eclipse-3.0.1-Test\workspace\Database\hibernate\com\omondo\hibernate\SalaryHistory.java
2004-10-07 15:39:34 No properties has been marked as being used in equals/hashcode for SalaryHistory. Using object identifier which is RARELY safe to use! See http://
2004-10-07 15:39:34 Writing D:\Outils\eclipse-3.0.1-Test\workspace\Database\hibernate\com\omondo\hibernate\Project.java
2004-10-07 15:39:34 Writing D:\Outils\eclipse-3.0.1-Test\workspace\Database\hibernate\com\omondo\hibernate\Employee.java
2004-10-07 15:39:34 Writing D:\Outils\eclipse-3.0.1-Test\workspace\Database\hibernate\com\omondo\hibernate\ProjDeptBudget.java
2004-10-07 15:39:34 No properties has been marked as being used in equals/hashcode for ProjDeptBudget. Using object identifier which is RARELY safe to use! See ht
2004-10-07 15:39:34 Writing D:\Outils\eclipse-3.0.1-Test\workspace\Database\hibernate\com\omondo\hibernate\Country.java

```

Extending EclipseDatabase

1. [Introduction](#)
2. [Database Definition](#)
3. [Database Definition Update](#)
4. [New Database Definition](#)
 1. [name](#)
 2. [label](#)
 3. [jdbcClass](#)
 4. [jdbcURL](#)
 5. [jdbcLevel](#)
 6. [torqueAdapter](#)
 7. [objPlatform](#)

1. Introduction

EclipseDatabase comes with a set of predefined known databases.

However, users should be able to

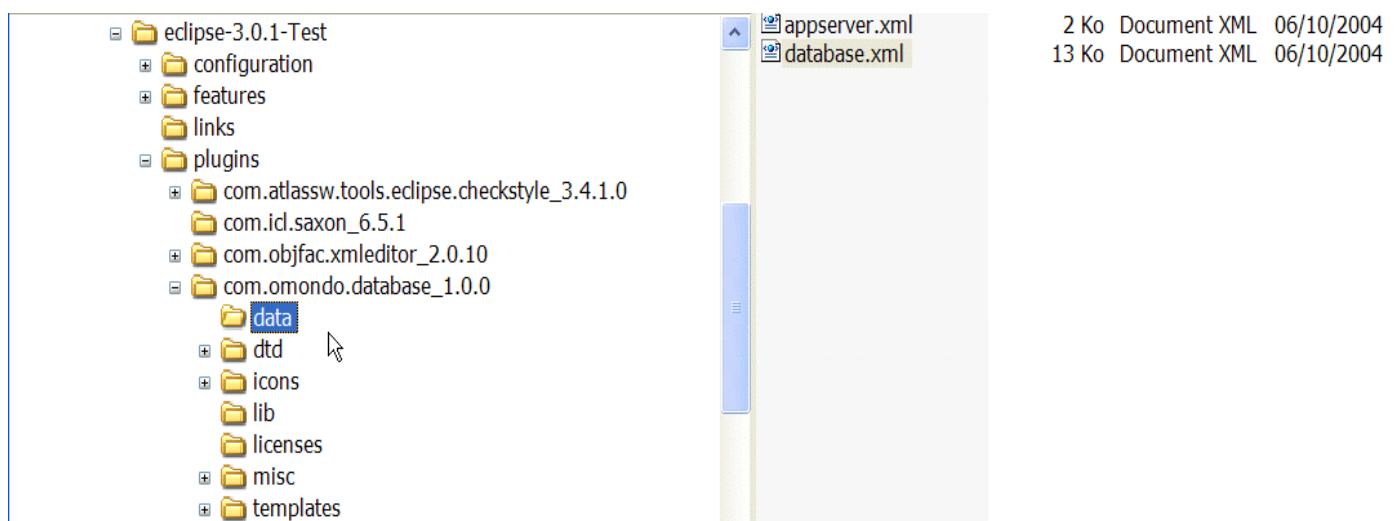
- Update an existing Database definition
- Add a new Database definition

Once you add a new Database definition, you will see the way to add SQL templates to be used with the following wizards :

- [Database Create SQL Script](#)
- [Database Schema SQL Script](#)

2. Database Definition

In the **%ECLIPSE_HOME%/plugins/com.omondo.database_x.y.z/data** directory



you should see a file called

database.xml

```

<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<databases xmlns="http://www.omondo.com/database/definition">
  <database
    name="msaccess"
    label="Access"
    jdbcClass="sun.jdbc.odbc.JdbcOdbcDriver"
    jdbcURL="jdbc:odbc:&#60;alias-name&#62;:"
    jdbcLevel="1.0"
    torqueAdapter="msaccess"
    ojbPlatform="MsAccess"
    hibernateClass="net.sf.hibernate.dialect.GenericDialect"
  />
  <database
    name="axion"
    label="Axion"
    jdbcClass="org.axiondb.jdbc.AxionDriver"
    jdbcURL="jdbc:axiondb:&#60;database_name&#62;[:&#60;database-directory&#62;:]"
    jdbcLevel="2.0"
    torqueAdapter="axion"
    ojbPlatform="Axion"
    hibernateClass="net.sf.hibernate.dialect.GenericDialect"
  />
  <database
    name="cloudscape"
    label="Cloudscape"
    jdbcClass="COM.cloudscape.core.JDBCDriver"
    jdbcURL="jdbc:cloudscape:&#60;full-db-path&#62;:"
    jdbcLevel="1.0"
    torqueAdapter="cloudscape"
    ojbPlatform=""
    hibernateClass="net.sf.hibernate.dialect.GenericDialect"
  />
  <database
    name="db2"
    label="DB2"
    jdbcClass="com.ibm.db2.jcc.DB2Driver"
    jdbcURL="jdbc:db2://&#60;host&#62;[:&#60;port&#62;]/&#60;database_name&#62;:"
    jdbcLevel="1.0"
    torqueAdapter="db2"
    ojbPlatform="Db2"
    hibernateClass="net.sf.hibernate.dialect.DB2Dialect"
  />

```

3. Database Definition Update

Most of the time when you want to modify an existing Database definition you need to modify either its **jdbcClass** or its **jdbcURL** definition.

Other values should remain untouched, otherwise strange side effects could occur.

Edit the **database.xml** file and patch the needed values.

Restart your Eclipse Platform to see the updated values.

4. New Database Definition

If you want to add a new Database definition, cut and paste an existing Database definition and modify the needed values.

Attribute values should be encoded with numerical entities.

4.1. name

Name is the internal EclipseDatabase key.

This value should be unique among Database definitions.

4.2. label

Label is the EclipseDatabase displayed label.

This value should be unique for a better understanding.

4.3. jdbcClass

jdbcClass is the corresponding JDBC Java Class.

4.4. jdbcURL

jdbcURL is the corresponding JDBC Connection URL pattern.

4.5. jdbcLevel

jdbcLevel is the corresponding JDBC Driver level.

Valid values should be :

- 1.0
- 2.0
- 3.0

This value is used with the [OJB Code Generator](#).

4.6. torqueAdapter

torqueAdapter is the name used to discriminate [Templates](#).

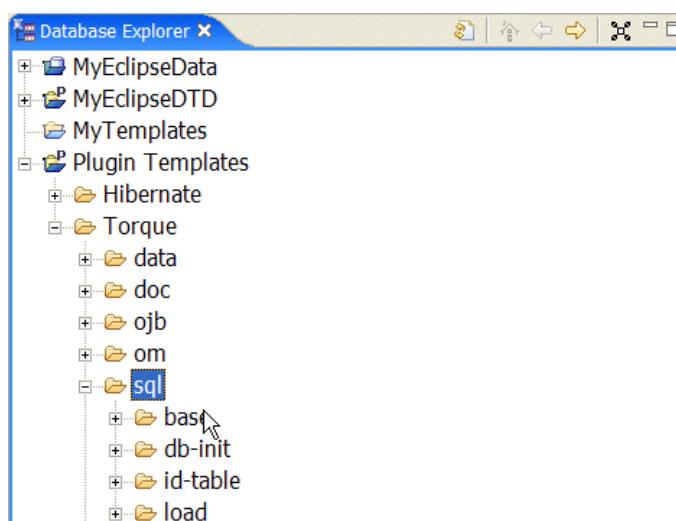
Below your [Plugin Templates](#) or [MyTemplates](#) you should see the following directories.

[Torque/sql/base](#)

[Torque/sql/db-init](#)

[Torque/sql/id-table](#)

[Torque/sql/load](#)



4.7. ojbPlatform

ojbPlatform is the targeted OJB Platform when you use the [Object Relational Bridge Resources wizard](#).

The [DatabaseExplorer](#) provides a shortcut through [MyEclipseDTD](#) to access the OJB's DTD.

In the Object Relational Bridge DTD you should see the valid platform values :

```
<!ATTLIST jdbc-connection-descriptor
  jcd-alias CDATA #REQUIRED
  default-connection (true | false) "false"
  platform (Db2 | Hsqldb | Informix | MsAccess | MsSQLServer |
             MySQL | Oracle | PostgreSQL | Sybase | SybaseASE |
             SybaseASA | Sapdb | Firebird | Axion | NonstopSql |
             Oracle9i ) "Hsqldb"
  jdbc-level (1.0 | 2.0 | 3.0) "1.0"
  eager-release (true | false) "false"
  batch-mode (true | false) "false"
  useAutoCommit (0 | 1 | 2) "1"
  ignoreAutoCommitExceptions (true | false) "false"

  jndi-datasource-name CDATA #IMPLIED

  driver CDATA #IMPLIED
  protocol CDATA #IMPLIED
  subprotocol CDATA #IMPLIED
  dbalias CDATA #IMPLIED
```

Hidden Options

1. [Introduction](#)
2. [.options](#)
3. [Debug Mode](#)
4. [Database Explorer](#)

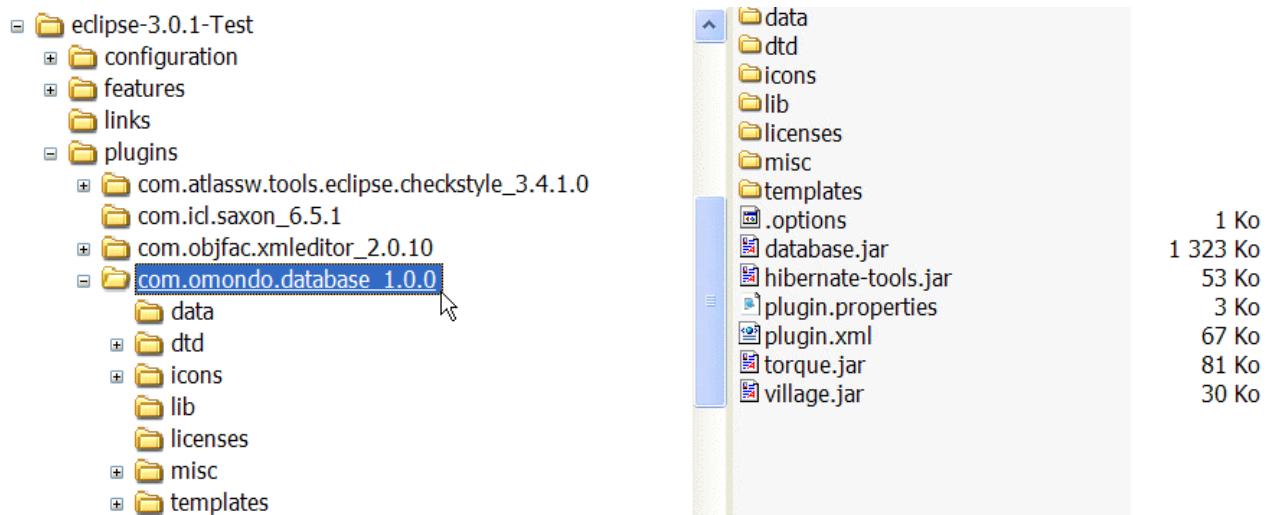
1. *Introduction*

EclipseDatabase can be switched into debug mode.

To switch the EclipseDatabase plugin into debug mode the **.options** and its properties are available.

2. .options

In the **%ECLIPSE_HOME%/plugins/com.omondo.database_x.y.z** directory



you should see a file called

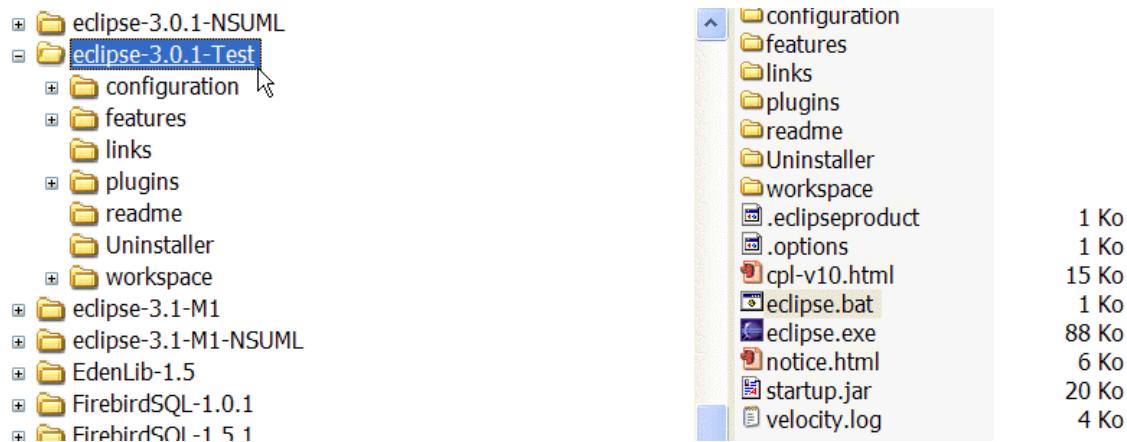
.options

```
com.omondo.database/debug=true
com.omondo.database/debug/explorer=true
com.omondo.database/debug/error=true
com.omondo.database/debug/warning=true
com.omondo.database/debug/info=true
com.omondo.database/debug/debug=true
com.omondo.database/debug/trace=true
com.omondo.database/debug/fatal=true
```

Values can be set to **true** or **false**.

This file is generally delivered at the root level of each plugin.

However users should copy or merge this file at the **%ECLIPSE_HOME%** root level.



Under Windows, Eclipse could be started in debug mode with the following statements of an hypothetical **eclipse.bat** file :

```
set ECLIPSE_HOME=D:\Outils\eclipse-3.0.1-Test
set PATH=%ECLIPSE_HOME%;%JAVA_HOME%\bin
start "Eclipse 3.0.1-Test" eclipse -debug -vmargs -Xms96m -Xmx512m
```

Starting Eclipse Platform this way, Eclipse will start itself in debug mode.

The **.options** file content will be read and loaded.

3. Debug Mode

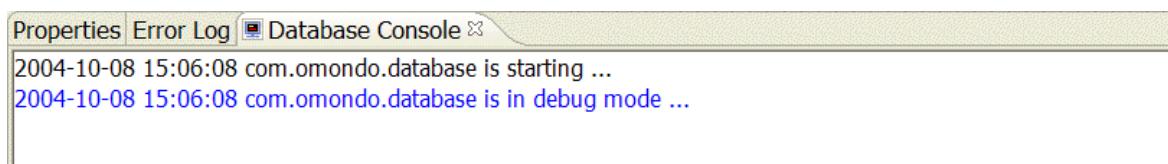
The properties

com.omondo.database/debug

is the most important key.

If set, the other properties will be analysed, otherwise they are simply ignored.

When set to true, EclipseDatabase starts in debug mode :



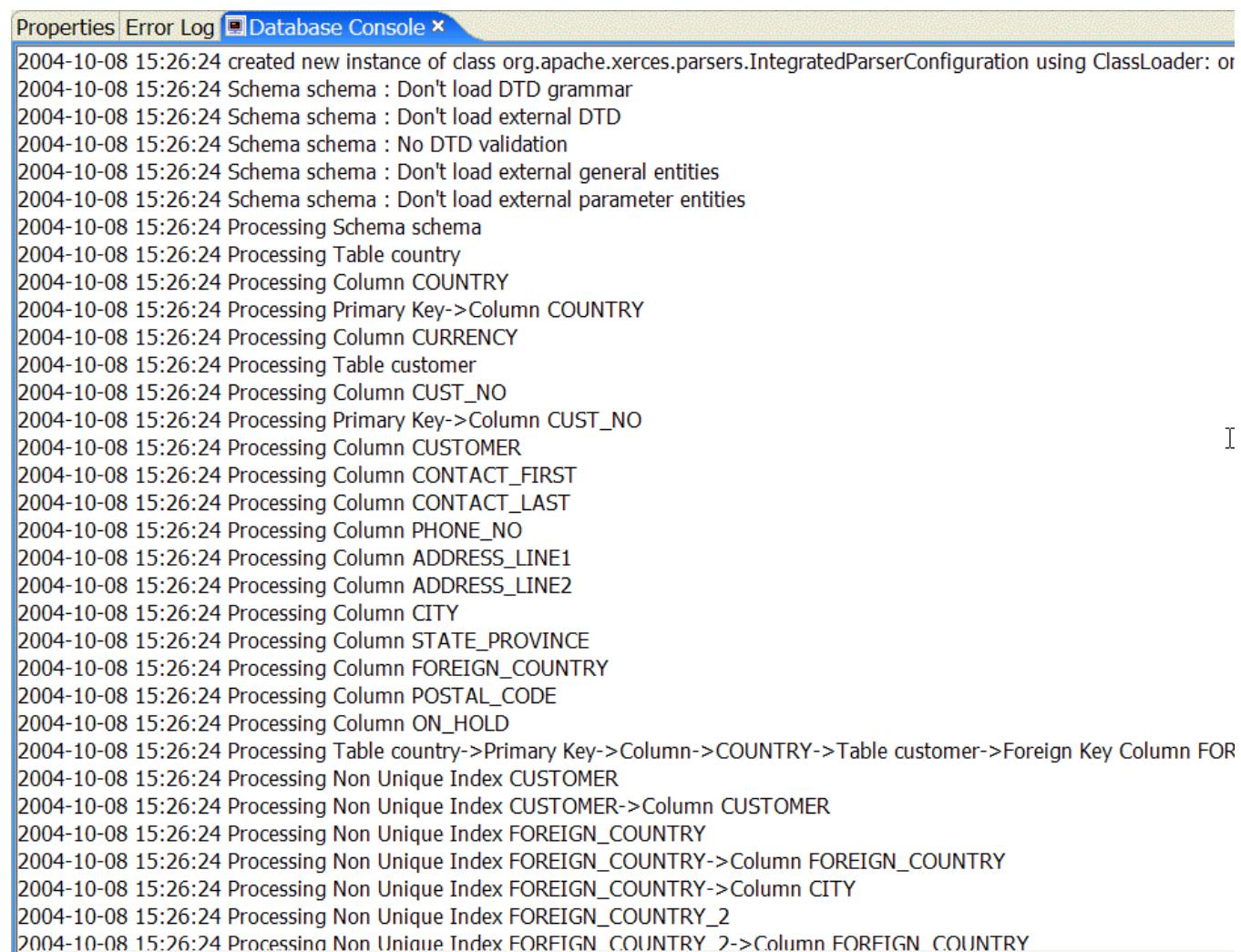
The following properties set various debug options.

com.omondo.database/debug/error
com.omondo.database/debug/warning
com.omondo.database/debug/info
com.omondo.database/debug/debug
com.omondo.database/debug/trace
com.omondo.database/debug/fatal

EclipseDatabase mostly uses the **info** property.

The other properties are used by external tools, like Velocity.

Here is a snapshot of the DatabaseConsole when a DatabaseConnection is read.



The screenshot shows a window titled "Database Console" with three tabs: "Properties", "Error Log", and "Database Console". The "Database Console" tab is selected and displays a log of database processing events. The log entries are as follows:

```

2004-10-08 15:26:24 created new instance of class org.apache.xerces.parsers.IntegratedParserConfiguration using ClassLoader: or
2004-10-08 15:26:24 Schema schema : Don't load DTD grammar
2004-10-08 15:26:24 Schema schema : Don't load external DTD
2004-10-08 15:26:24 Schema schema : No DTD validation
2004-10-08 15:26:24 Schema schema : Don't load external general entities
2004-10-08 15:26:24 Schema schema : Don't load external parameter entities
2004-10-08 15:26:24 Processing Schema schema
2004-10-08 15:26:24 Processing Table country
2004-10-08 15:26:24 Processing Column COUNTRY
2004-10-08 15:26:24 Processing Primary Key->Column COUNTRY
2004-10-08 15:26:24 Processing Column CURRENCY
2004-10-08 15:26:24 Processing Table customer
2004-10-08 15:26:24 Processing Column CUST_NO
2004-10-08 15:26:24 Processing Primary Key->Column CUST_NO
2004-10-08 15:26:24 Processing Column CUSTOMER
2004-10-08 15:26:24 Processing Column CONTACT_FIRST
2004-10-08 15:26:24 Processing Column CONTACT_LAST
2004-10-08 15:26:24 Processing Column PHONE_NO
2004-10-08 15:26:24 Processing Column ADDRESS_LINE1
2004-10-08 15:26:24 Processing Column ADDRESS_LINE2
2004-10-08 15:26:24 Processing Column CITY
2004-10-08 15:26:24 Processing Column STATE_PROVINCE
2004-10-08 15:26:24 Processing Column FOREIGN_COUNTRY
2004-10-08 15:26:24 Processing Column POSTAL_CODE
2004-10-08 15:26:24 Processing Column ON_HOLD
2004-10-08 15:26:24 Processing Table country->Primary Key->Column->COUNTRY->Table customer->Foreign Key Column FOR
2004-10-08 15:26:24 Processing Non Unique Index CUSTOMER
2004-10-08 15:26:24 Processing Non Unique Index CUSTOMER->Column CUSTOMER
2004-10-08 15:26:24 Processing Non Unique Index FOREIGN_COUNTRY
2004-10-08 15:26:24 Processing Non Unique Index FOREIGN_COUNTRY->Column FOREIGN_COUNTRY
2004-10-08 15:26:24 Processing Non Unique Index FOREIGN_COUNTRY->Column CITY
2004-10-08 15:26:24 Processing Non Unique Index FOREIGN_COUNTRY_2
2004-10-08 15:26:24 Processing Non Unique Index FOREIGN_COUNTRY_2->Column FOREIGN_COUNTRY

```

4. Database Explorer

The properties

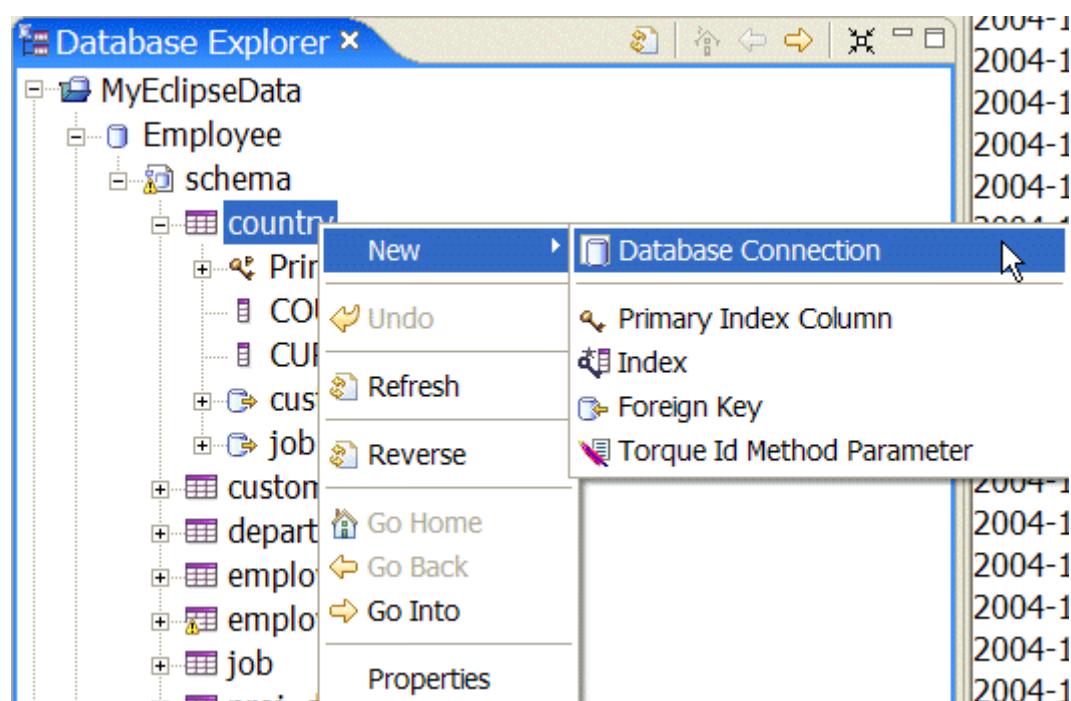
com.omondo.database/debug/explorer

activates [MyEclipseData](#) in read/write mode.

This function is just a technology preview of what you will see in the workspace.

The MyEclipseData doesn't belong to the workspace and as such doesn't have an associated Database Diagram.

However you can work in a tree mode on your Database Structure.



New available Commands :

Database Table

- New->Primary Index Column
- New->Index
- New->Foreign Key
- New->Torque Id Method Parameter

Database Primary Index

- New->Primary Index Column

Primary Index Column

- Move Up
- Move Down

Index

- New->Index Column

Index Column

- Move Up
- Move Down

Foreign Key

- New->Foreign Key Column

Properties are also accessible in read/write mode.

