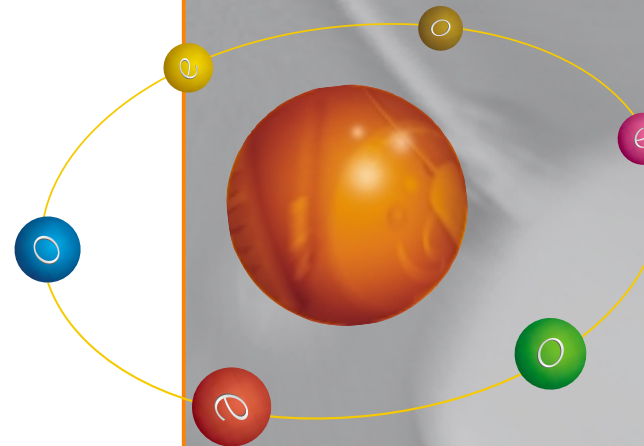


Objectteering/UML

Objectteering/EJB User Guide

Version 5.2.2



www.objectteering.com

Objectteering
Software

Taking object development one step further

Information in this document is subject to change without notice and does not represent a commitment on the part of Objectteering Software. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license agreement. The purchaser may make one copy of the software for backup purposes. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or information storage and retrieval systems, for any purpose other than the purchaser's personal use, without the express written consent of Objectteering Software.

© 2003 Objectteering Software

Objectteering/UML version 5.2.2 - CODOBJ 001/003

Objectteering/UML is a registered trademark of Objectteering Software.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

UML and OMG are registered trademarks of the Object Management Group. Rational ClearCase is a registered trademark of Rational Software. CM Synergy is a registered trademark of Telelogic. PVCS Version Manager is a registered trademark of Merant. Visual SourceSafe is a registered trademark of Microsoft. All other company or product names are trademarks or registered trademarks of their respective owners.

Contents

Chapter 1: Introduction	
Overview of Objectteering/EJB	1-3
Structure of the Objectteering/EJB user guide	1-5
Developing EJB components	1-6
Glossary	1-7
Chapter 2: Using the Objectteering/EJB module	
Working with the Objectteering/EJB module	2-3
Chapter 3: First steps	
Overview of the Objectteering/EJB first steps	3-3
Initializing the first steps project	3-4
Converting a class to an EJB	3-13
Creating and deploying the EJB JAR file	3-23
Chapter 4: Principles of Objectteering/EJB	
Selecting and parameterizing the Objectteering/EJB module	4-3
Work products	4-4
Tagged values, notes and stereotypes	4-5
Consistency checks	4-6
Chapter 5: Converting classes into EJBs and correspondence between EJB parts	
Converting a class into an EJB	5-3
Commands used to update EJBs	5-4
Chapter 6: Objectteering/EJB functions	
Package functions	6-3
Subsystem functions	6-8
Subsystem and package functions	6-13
Class functions	6-15
Operation functions	6-20
Tagged value types	6-23
Note types	6-26
Stereotypes	6-27

Chapter 7: Parameterization	
Defining module parameters	7-3
Parameter sets	7-4
Chapter 8: Specific EJB properties	
WebLogic 7.0 and Objectteering/UML	8-3
WebLogic 6.0 and Objectteering/UML	8-4
WebSphere 3.5 and Objectteering/UML	8-22
WebSphere 4.0 and Objectteering/UML	8-23
IONA iPortal Application Server 3.0 and Objectteering/UML	8-24
Sun J2EE Server and Objectteering/UML	8-25
JBoss 3.0 and other application servers	8-26
Index	

Chapter 1: Introduction

Overview of Objectteering/EJB

Welcome to the *Objectteering/EJB* module!

The *Objectteering/EJB* module is used to generate EJB components. Furthermore, during the modeling phase, a reverse feature is provided to enable you to use existing EJB components.

The Objectteering/EJB module is aimed at developers who wish to accelerate the creation of EJB components. You must be familiar with the Objectteering/UML environment, detailed in the following user guides:

- ◆ *Objectteering/Administrating Objectteering Sites*
- ◆ *Objectteering/UML Modeler*

Chapter 3 of this user guide, "*First Steps*", provides the user with a step by step demonstration of the workings of the *Objectteering/EJB* module. We also recommend that all users try out the first steps project in the *Objectteering/Introduction* user guide, in order to become sufficiently at ease with the various general functions provided by Objectteering/UML.

Functions

Using the *Objectteering/EJB* module, it is possible to:

- ◆ create a new EJB 1.1 or 2.0
- ◆ convert a UML class to an EJB 1.1 or 2.0
- ◆ check the validity of your EJB model
- ◆ create a platform independent JAR file and deploy it on your application server
- ◆ reverse an existing EJB
- ◆ create standard modeling of your EJBs and then use specific functions for your application server

Parameterization

Objectteering/EJB can be configured to specify:

- ◆ default inheritance
- ◆ the application server path and options
- ◆ the packaging tool command line
- ◆ reverse options
- ◆ the validation log file path
- ◆ external edition options
- ◆ IONA iPortal application server options

Model-driven generation

Unlike other CASE tools, *Objectteering/UML* entirely generates an EJB application, from the model, textual specifications and tagged values right up to the executables. Therefore, the user need develop nothing in EJB outside the *Objectteering/UML* tool. With *Objectteering/UML*, you can enter everything directly into the CASE tool, modify generated source zones and present the model sections which correspond to the generated code. In this way, *Objectteering/UML* ensures consistency at all times between the model and code.

Structure of the Objectteering/EJB user guide

The *Objectteering/EJB* user guide is structured as follows:

- ◆ Chapter 1 - Introduction to the *Objectteering/EJB* module
 - ◆ Chapter 2 - Using the *Objectteering/EJB* module
 - ◆ Chapter 3 - First Steps
 - ◆ Chapter 4 - Principles of *Objectteering/EJB*
 - ◆ Chapter 5 - Converting classes into EJBs and correspondence between EJB parts
 - ◆ Chapter 6 - *Objectteering/EJB* functions
 - ◆ Chapter 7 - Parameterization
 - ◆ Chapter 8 - Specific EJB properties
-

Developing EJB components

The development of an EJB component consists of:

- ◆ creating a UML model
- ◆ creating a JAR file
- ◆ deploying the JAR file on your application server

The *Objectteering/EJB* module provides a set of stereotypes and tagged values specific to EJB which can be used to annotate model elements.

Stereotypes are used to define a specific EJB element. For example, the EJB <<EJBImplementation>> stereotype designates an implementation class of an EJB.

Tagged values inform the generator of the implementation rules applied. For example, the {EJBSessionType} tagged value defines the type of the EJB. This tagged value can take "Stateful" or "Stateless" as its parameter.

For further information and a detailed description of these annotations, please refer to chapter 5 of this user guide, "*Objectteering/EJB functions*".

Glossary

- ◆ *EJB*: Software component designed to provide remote services.
 - ◆ *Session Stateful*: EJB capable of providing a service based on a conversation with the client.
 - ◆ *Session Stateless*: EJB providing a simple service, which does not need to maintain a conversational state.
 - ◆ *Entity*: EJB whose attributes are mapped to an RDBMS. This EJB represents a persistent object.
 - ◆ *Deployer tool*: Software provided with the application server. This tool configures and generates the EJB's container. This is the last step in the creation of an EJB.
 - ◆ *Security role*: User group with the same rights regarding the calling of a function.
 - ◆ *Container*: Part of an EJB generated by the application server, to provide the EJB with runtime services.
 - ◆ *Home interface*: Interface defining the lifecycle of an EJB (creation, search, and so on).
 - ◆ *Remote interface*: Interface defining methods exported by the EJB, in order to access it.
 - ◆ *Bean class*: Class containing the executable part of the EJB, implementation of the remote methods, and initializations automatically called when an instance is created by the home.
 - ◆ *Primary key class*: Tool class to find an instance of an EJB from its database keys.
-

Chapter 2: Using the Objectteering/EJB module

Working with the Objectteering/EJB module

Prerequisites

In order to work with the *Objectteering/EJB* module, the *Objectteering/Java* module version 2.1 (provided with Objectteering V5.1.1) or higher must also be present.

The SDK Standard Edition (version 1.2.2 or higher) and the SDK Enterprise Edition (version 1.2.1 or higher) must already have been installed. You can install these while proceeding with the installation of Objectteering/UML.

You must have the correct license in order to be able to use the *Objectteering/EJB* module.

Note: <\$OBJING_PATH> designates the Objectteering/UML installation directory (for example, C:\Program Files\Objectteering).

Installing the Objectteering/EJB module

The *Objectteering/EJB* module can be delivered to and installed on your Objectteering/UML site automatically during the Objectteering/UML installation procedure, by selecting "*Custom*" installation and carrying out the necessary steps. For further information, please refer to chapter 2 of the *Objectteering/Introduction* user guide.

Installation directories

Module data is installed in the \$OBJING_PATH/modules/EJB directory. This directory contains:

- ♦ bin
- ♦ doc
- ♦ externalization
- ♦ FirstStepsContainer
- ♦ res

Using the module

In order to use the *Objectteering/EJB* module, you simply have to:

- ◆ create a UML modeling project
- ◆ select the module for the current UML modeling project

Note: When the *Objectteering/EJB* module is selected, a window asking for the name and location of the EJB platform you will be using appears.

Creating a UML modeling project for developing EJB applications

For information on how to create a UML modeling project, please refer to "Creating a UML modeling project" section in chapter 2 of the *Objectteering/UML Modeler* user guide.

Selecting the Objectteering/EJB module for the new UML modeling project

To select the *Objectteering/EJB* module, simply carry out the steps illustrated below (Figure 2-1).

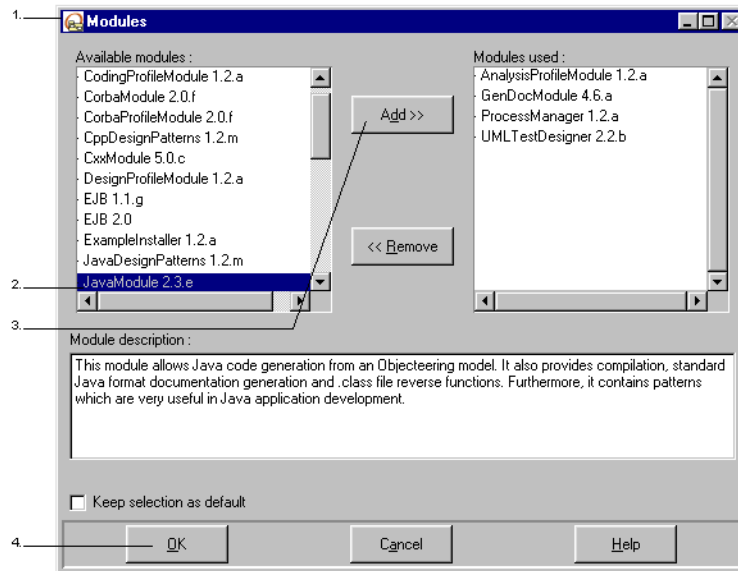



Figure 2-1. Selecting the *Objectteering/Java* module

Steps:

- 1 - Select the "Modules..." command in the "Tools" menu or click on the  "UML modeling project modules" icon to launch the "Modules" window.
- 2 - Select the Java module from the available modules list on the left-hand side.
- 3 - Click on the "Add" button. The Java module then appears in the right-hand "Modules used" column.
- 4 - Click on "OK" to confirm. If the "Keep selection as default" box is checked, the Java module will automatically be available during future Objectteering/UML sessions.

Once the Java module has been selected, perform the same steps to select the EJB module. After selecting the *Objectteering/EJB* platform, the window shown in Figure 2-2 then appears.

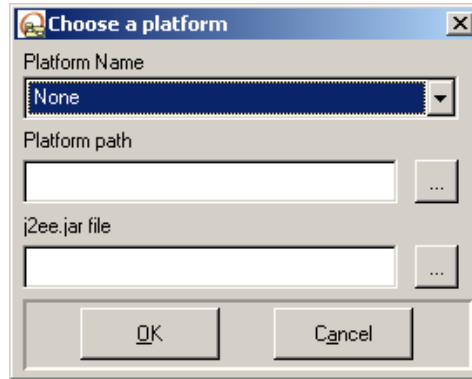



Figure 2-2. The window for selecting your EJB platform

Simply click on the  to open the dropdown list of possible platforms, and select the one relevant to you. Then indicate the location of this platform in the "*Platform path*" field. Finally, select the j2ee.jar file to add to your CLASSPATH. If you want to create an EJB 1.1, use the j2ee provided with j2eesdk 1.2, whilst if you want to create an EJB 2.0, use the j2ee provided with j2eesdk 1.3.

Chapter 3: First steps

Overview of the Objectteering/EJB first steps

The *Objectteering/EJB* module first steps present an EJB demonstration project, designed to help you discover the different features of the *Objectteering/EJB* module, step by step.

We recommend that before starting every user carry out the general Objectteering/UML first steps in the *Objectteering/Introduction* user guide.

The *Objectteering/EJB* first steps will demonstrate:

- ◆ how to initialize the First Steps project
- ◆ how to convert an existing class into a session stateless EJB
- ◆ how to generate the Jar file containing the EJB

The creation of a new EJB skeleton will be detailed in the sections which follow.

We are going to create a simple session stateless EJB which calculates the price inclusive of tax from the price before tax.

Initializing the first steps project

Preparation steps

Before starting work with the *Objectteering/EJB* module in a UML modeling project, you must first prepare the working environment, by carrying out the following steps.

- 1 - Launch *Objectteering/UML Modeler*.
- 2 - Create a new UML modeling project through the "*File/New...*" command.
- 3 - Select the *Objectteering/Java* module.
- 4 - Select the *Objectteering/EJB* module.

Note: After selection of the *Objectteering/EJB* module, two packages are added to your model (the "*java*" and "*javax*" packages), which are necessary to the correct functioning of the *Objectteering/EJB* module.

Importing a model into the first steps project

We are now going to import the "*EJBFirstSteps*" demonstration project, which will be our model, into our newly created UML modeling project. To do this, simply carry out the steps illustrated in Figure 3-1.

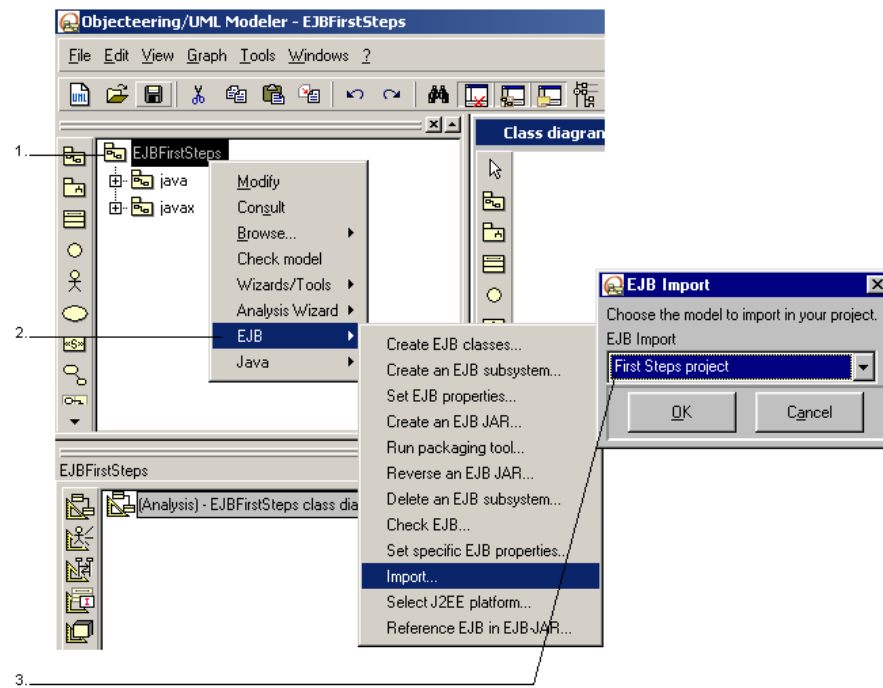


Figure 3-1. Importing the EJB first steps project

Steps:

- 1 - Select the UML model root of your new UML modeling project by right-clicking on the root package.
- 2 - Run the "*EJB/Import*" commands. The "*EJB Import*" window then appears.
- 3 - Select the "*First Steps project*", and click on "OK" to confirm.

As can be seen in Figure 3-2, this operation creates four packages.

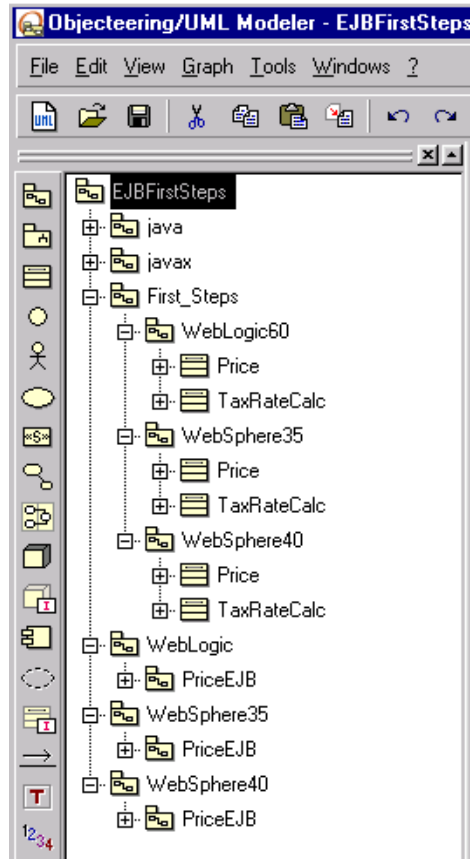


Figure 3-2. The result of importing the EJB first steps project

The new packages are as follows:

- ◆ the "*First_Steps*" package, which contains the classes that you will convert to an EJB
- ◆ the "*WebLogic*" package, which contains the final EJB ready for deployment on an application server
- ◆ the "*WebSphere35*" package, which contains the final EJB ready for deployment on an application server
- ◆ the "*WebSphere40*" package, which contains the final EJB ready for deployment on an application server

Configuring the Objectteering/EJB module

After selection of the module, the following window appears (Figure 3-3).

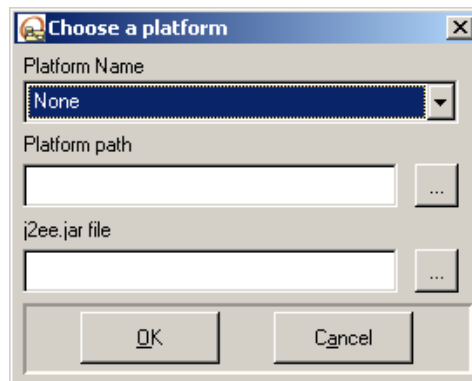



Figure 3-3. The "Choose a platform" window

Simply choose the application server you are going to use and indicate its path.
If the window shown in Figure 3-3 does not appear, carry out the steps detailed below.

The  "Modify module parameter configuration" icon or the "Tools/Modify configuration..." menu is used to open the "Modifying configuration" dialog box.

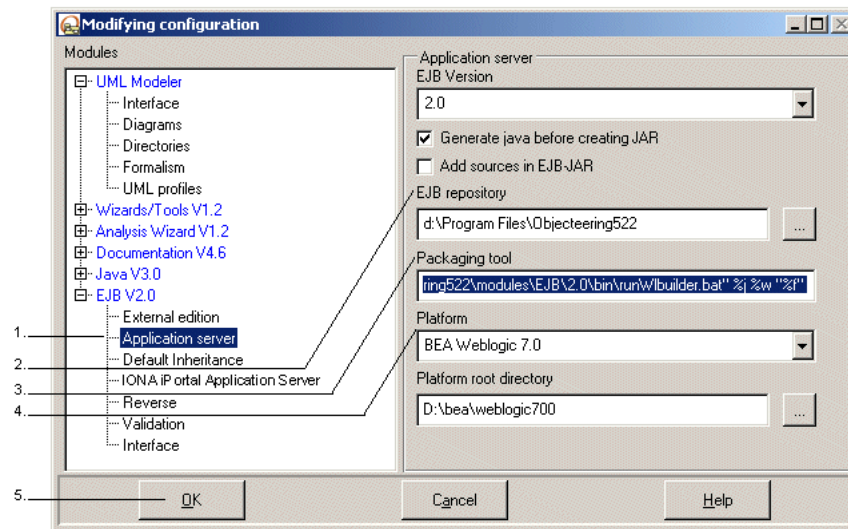


Figure 3-4. Editing the configuration of the application server

Steps:

- 1 - Select the "Application server" sub-category in the EJB section.
- 2 - Indicate the complete EJB repository path, which is where the *Objectteering/EJB* module will save your EJB.
- 3 - Indicate the complete path of the deployer tool, for example, C:\deployer.bat %f. Check that the installation path of the application server is connected to the explorer. The module replaces the "%f" string with the name of the Jar file which is to be deployed and the %6 by the name of the Jar file built. This tool will be launched by the "Run packaging tool" command.
- 4 - Select the application server you wish to use.
- 5 - Confirm.

Note: The deployer tool is application server specific.

Chapter 3: First steps

We are now going to set the validation log file path, which is useful if you want to store the result of the validation command.

Note: This file is overwritten each time you run the validation command.

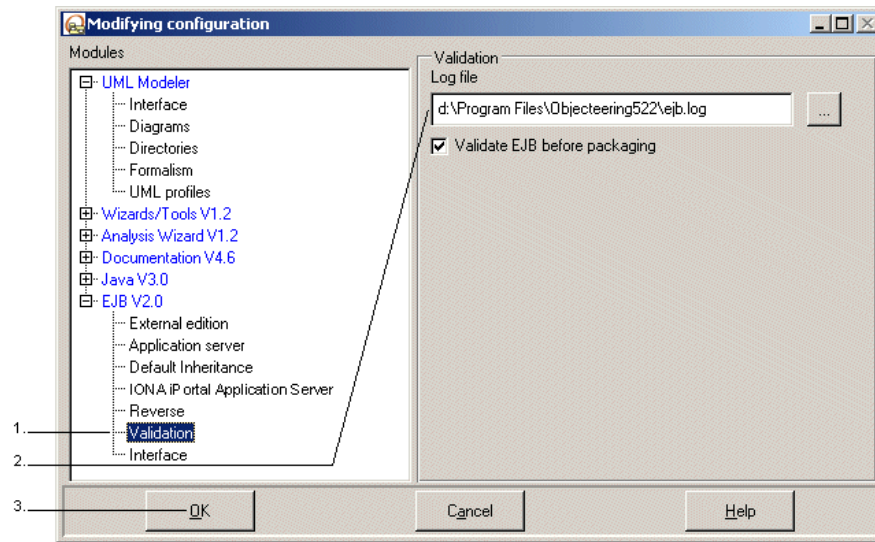


Figure 3-5. Editing the configuration of validation

Steps:

- 1 - Select the "Validation" sub-category in the EJB section.
- 2 - Indicate the complete path of the log file tool, for example, C:\Validation.log.
- 3 - Confirm.

Modifying an operation

The next step is to modify the "*getInitialContext*" operation in the "*TaxRateCalc*" class (as shown in Figure 3-6) for your application server.

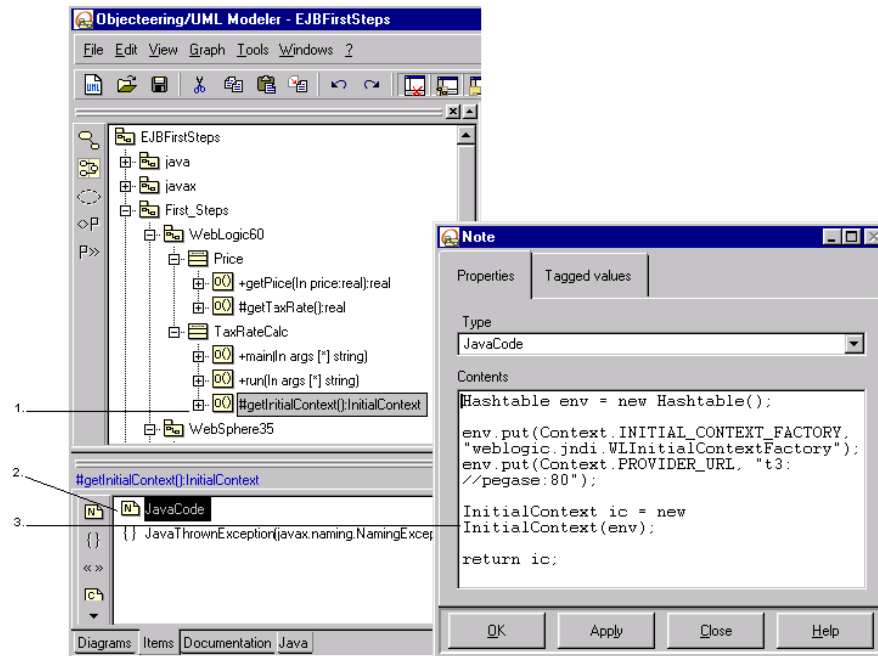


Figure 3-6. Modifying the Java code for the "*getInitialContext*" operation

Chapter 3: First steps

Steps:

- 1 - In the explorer, select the "*getInitialContext*" operation belonging to the "*TaxRateCalc*" class of the package corresponding to your application server (WebLogic6.0 for Weblogic and WebSphere3.5 for WebSphere).
 - 2 - Select the "*JavaCode*" note in the "*Items*" tab in the properties editor.
 - 3 - Make the following changes to the `env.put(Context.PROVIDER_URL, "t3://pegase:80");` line:
 - ◆ Replace "t3" with the protocol you have chosen (t3 or http for WebLogic and iiop for WebSphere).
 - ◆ Replace "pegase" with the hostname of your application server.
 - ◆ Replace "80" with the port number you have chosen (WebLogic default setting: 7001).
-

Converting a class to an EJB

Creating the EJB structure

We are now going to create an EJB structure from the existing class. The "Price" class already contains all the operations needed to create the EJB.

- ♦ the "getPrice (in price:real)" operation returns the price including tax of the "price" parameter.
- ♦ the "getTaxRate()" operation returns the tax rate stored in the server environment variable.

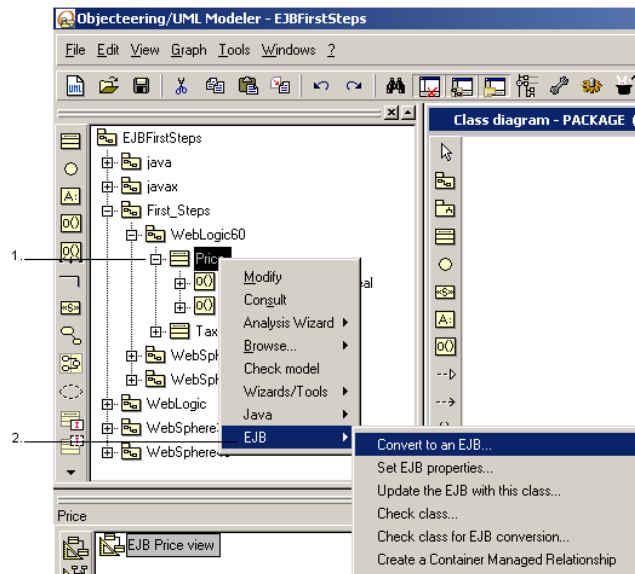


Figure 3-7. Running the "Convert to an EJB" command

Steps:

- 1 - Right-click on the "Price" class to open the context menu.
- 2 - Run the "EJB/Convert to an EJB" command.

This command opens the following dialog box (Figure 3-8).

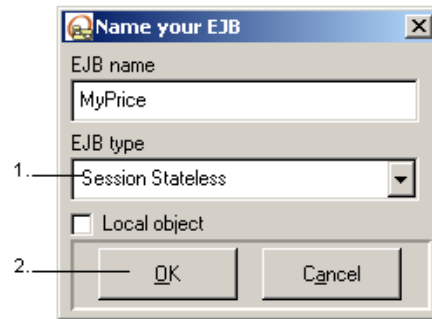


Figure 3-8. Entering essential information

Steps:

- 1 - Select the "*Session Stateless*" type from the combobox.
- 2 - Confirm by clicking on the "*OK*" button.

The window shown in Figure 3-9 then appears.

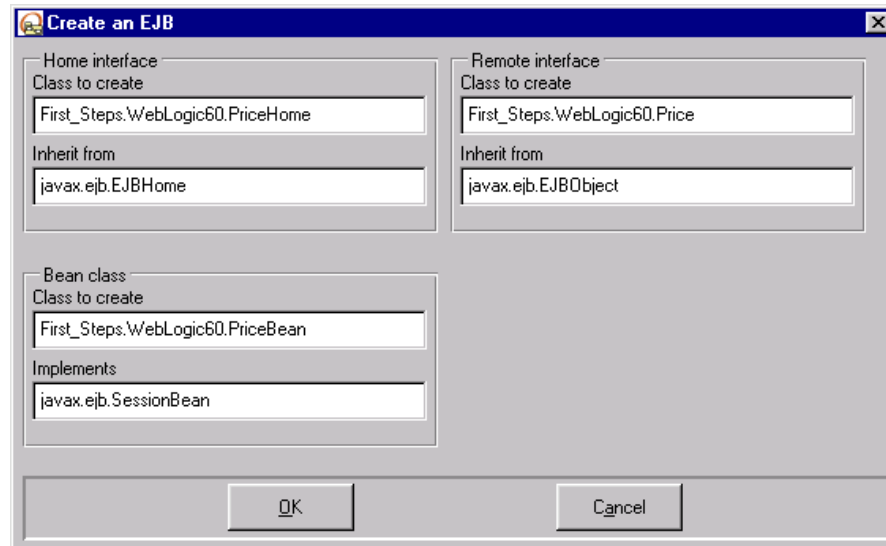


Figure 3-9. The "Create an EJB" window

Confirm by clicking on the "OK" button.

Creating an EJB subsystem

We are now going to create a subsystem, to reference the three classes of our EJB and to save EJB properties such as EJBEnvEntries which will contain our tax rate.

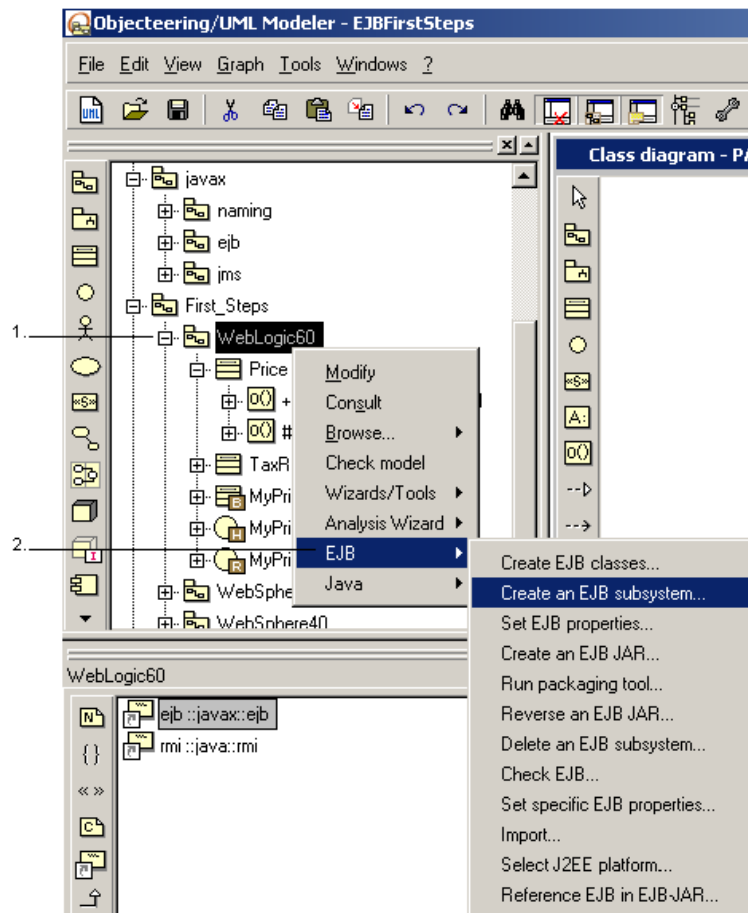


Figure 3-10. The first steps in creating an EJB subsystem

Steps:

- 1 - Right-click on the "WebLogic60" package to open the context menu.
- 2 - Run the "Create an EJB subsystem" command from the "EJB" menu, and carry out the steps shown in Figure 3-11.

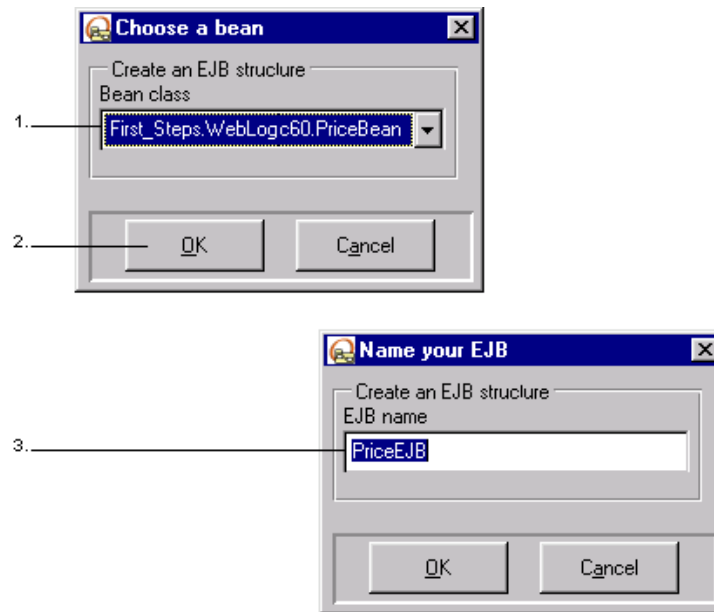


Figure 3-11. Choosing a bean and naming your EJB

Steps:

- 1 - Click on the arrow and select your bean class from the combobox list.
- 2 - Confirm by clicking on "OK".
- 3 - Enter a name for your EJB, and continue by carrying out the steps shown in Figure 3-12.

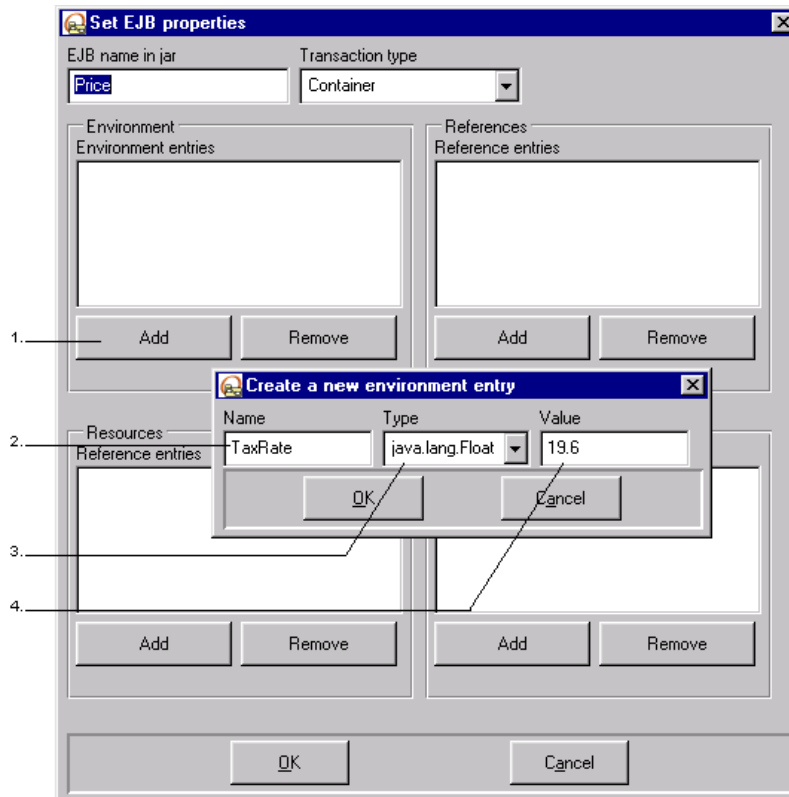


Figure 3-12. Creating a new environment entry

Steps:

- 1 - Click on the "Add" button. The "Create a new environment entry" window then appears.
- 2 - Enter the name of the variable.
- 3 - Click on the arrow to open the combobox list, and select the variable type.
- 4 - Enter the value, and click on "OK" to confirm.

The module will now generate the subsystem.

Creating a Java generation work product

In Objectteering, the Java generation work product provides the commands for generating code, compiling and generating documentation. It can also be used to administrate the files which have been produced. Thus, if you destroy the generation work product, you will also destroy the files produced. For further information on Java generation work products, please refer to the *Objectteering/Java* user guide.

Using the example shown in Figure 3-13, we are going to create a Java generation work product for the "WebLogic60" package.

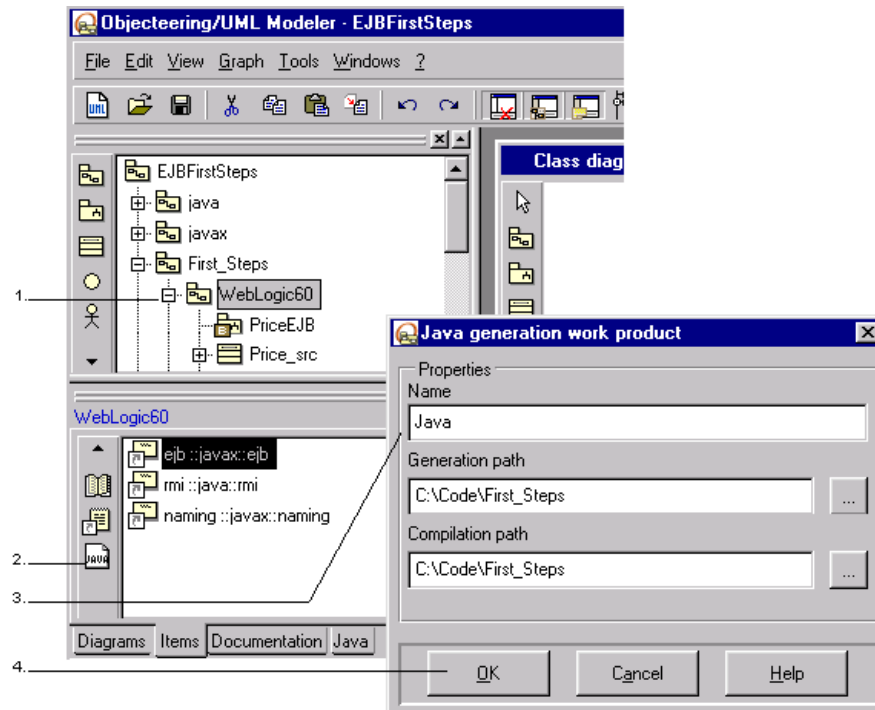


Figure 3-13. Creating a Java generation work product

Chapter 3: First steps

Steps:

- 1 - Select the "*WebLogic60*" package in the explorer.
- 2 - Click on the "*Java generation work product*" button in the "*Items*" tab of the properties editor.
- 3 - Give the same path for generation and compilation.
- 4 - Confirm, and continue by carrying out the steps shown in Figure 3-14.

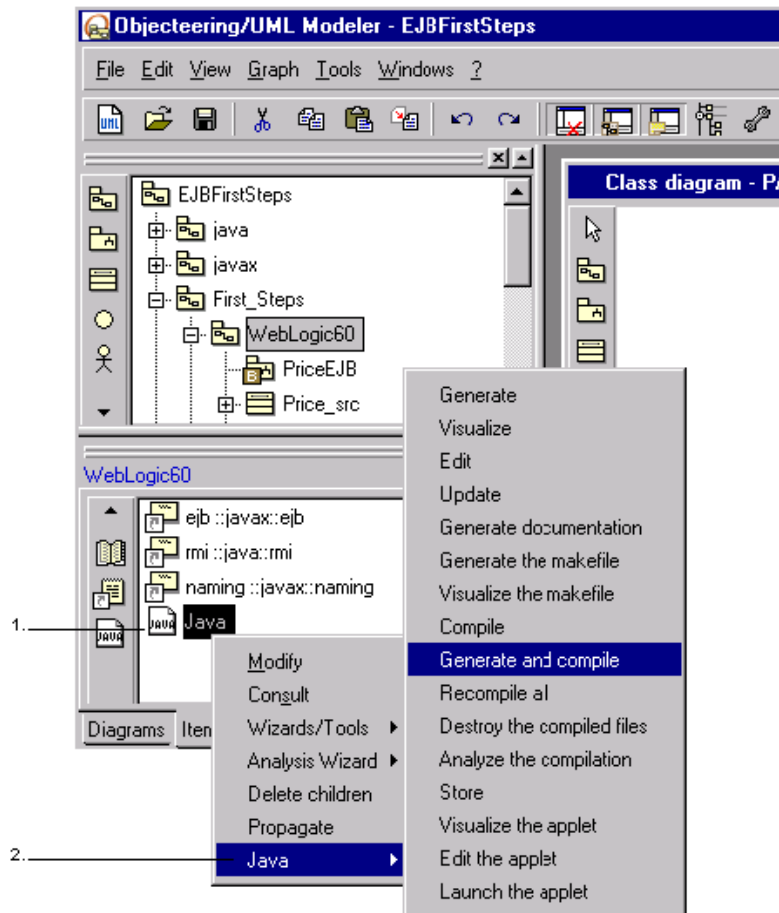


Figure 3-14. Running the "Generate and compile" command

Chapter 3: First steps

Steps:

- 1 - Right-click on the "*WebLogic60_Java*" Java generation work product in the "*Items*" tab of the properties editor, to open the context menu.
- 2 - Click on the "*Java*" option.
- 3 - Run the "*Generate and compile*" command.

Note: The purpose of a generation work product is to define generation options for the associated model elements. A Java generation work product can also be defined on elements other than the root package. A work product can be created on each package or class.

Note: Double-clicking on the generation work product executes the "*Generate*" command for the selected work product.

Creating and deploying the EJB JAR file

We are now going to create the JAR file which will contain all the files which make up the EJB.

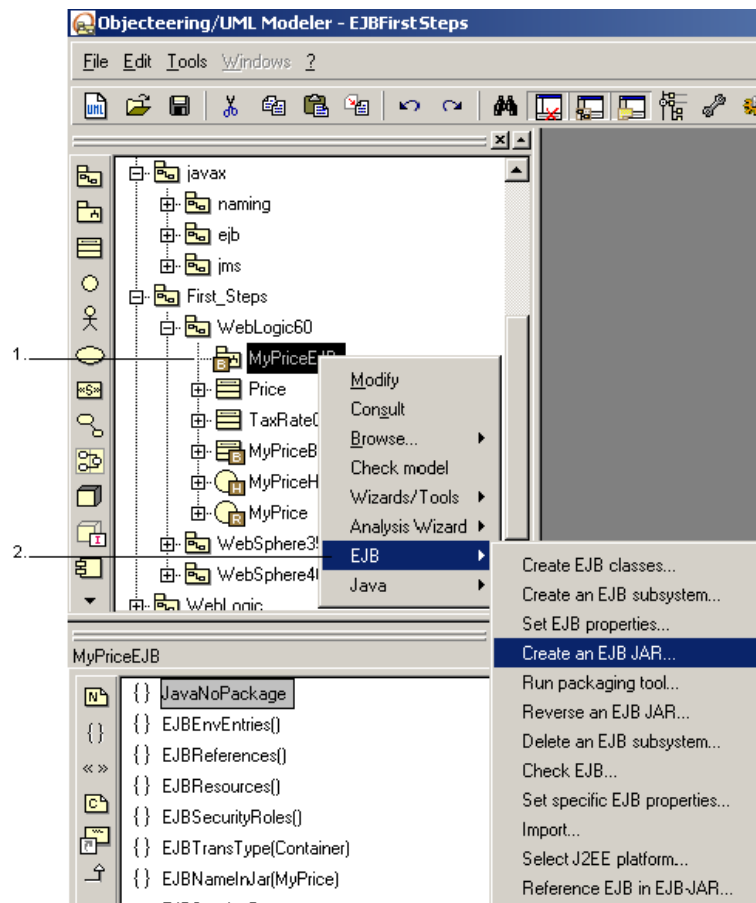


Figure 3-15. Creating the EJB JAR file

Steps:

- 1 - Right-click on the "PriceEJB" subsystem in the "First_Steps" package.
- 2 - Run the "EJB/Create an EJB JAR" command. The "Create a JAR from an EJB" window then appears (as shown in Figure 3-16).

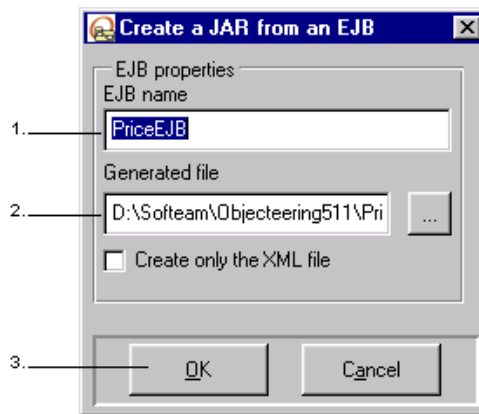


Figure 3-16. Creating a JAR file from an EJB

Steps:

- 1 - Enter a name for your EJB.
- 2 - Enter the path for the generated file.
- 3 - Confirm by clicking on "OK".

The module will generate the Java code which will be compiled, and then the JAR file will be created. This file is now ready to be deployed in the application server.

Deploying the EJB on your application server

To deploy your EJB on your application server you need to run its deployer tool. To do this, simply carry out the following steps:

- 1 - Right-click on the subsystem in the "*First_Steps*" package.
- 2 - Click on the "*Run packaging tool*" command in the "*EJB*" menu.
- 3 - Please refer to the documentation of your application server for the next steps.

You can now test your EJB by running the client file with the following command line:

```
java -classpath %CLASSPATH%;. First_Steps.TaxRateCalc 1000.
```

Don't forget to configure the *Objectteering/Java* module. For further details, please refer to the *Objectteering/Java* user guide.

Chapter 4: Principles of Objecteering/EJB

Selecting and parameterizing the Objectteering/EJB module

Selecting the Objectteering/EJB module

Before being able to use the *Objectteering/EJB* module in your UML modeling project, it must first be selected. This operation is detailed in the "*Selecting modules in the current UML modeling project*" section in chapter 3 of the *Objectteering/Introduction* user guide.

Parameterizing the Objectteering/EJB module

The *Objectteering/EJB* module provides various ways of customizing the EJB generator, in order to adapt it to your programming style:

- ◆ Parameterizing through general EJB module parameters (for further information, please refer to chapter 7, "*Parameterization*", of this user guide).
 - ◆ Parameterizing using tagged values, notes and stereotypes. By annotating the model with tagged values, notes and stereotypes, the user can define code generation more precisely (for further information, please refer to the "*Tagged values*", "*Notes*" and "*Stereotypes*" sections in chapter 6 of this user guide).
 - ◆ Parameterizing using the *UML Profile Builder* tool. Code generated can be adapted, by redefining the J methods in charge of producing zones of EJB code. Through *Objectteering/UML Profile Builder*, you may define your own tagged values, documents and items, as well as rules for checking, generation and validation (for further information, please refer to the *Objectteering/UML Profile Builder* user guide).
-

Work products

The *Objectteering/EJB* module is used to generate EJB in Java code from a UML model created in Objectteering/UML.

Before generating Java code, a work product must be created. Work products represent elements external to Objectteering/UML, which are delivered externally or which are used by other tools. They can be deliverables, such as documentation, source code (C++, Java, etc), database SQL schemas or any other element produced by Objectteering/UML. Work products can be accessed from the properties editor. They maintain consistency with related external elements (typically files) and provide services specific to the target and the external element, such as "*Generate*" or "*Visualize*".

Work products follow the project's composition structure (project/package/class logic). If you create a work product on a package during the first generation, a work product will be created for each component (sub-package or class) which may have such a work product. If components are added after the last generation, subsequent generation will create work products on the new components which may have such work products. In this way, these work products will maintain consistency with the model.

Tagged values, notes and stereotypes

Objectteering/UML is a multi-target workshop used to model a large quantity of application elements, whatever the computing language used.

However, during the technical designing and programming phases, certain implementation details expressed in the target language have to be specified and added to the model. This information, used to complete the model before generation, is entered using:

- ◆ *Tagged values*, which provide implementation rules for the generator
- ◆ *Notes*, which correspond to the zones inserted directly into the generated code
- ◆ *Stereotypes*, which provide implementation rules for the generator, different to those of a non-stereotyped element

Tagged values, notes and stereotypes specific to EJB can be added to a model element only if the *Objectteering/EJB* has been selected. For further information on tagged values, notes and stereotypes, please refer to the related sections in chapter 6 of this user guide.

Consistency checks

Objectteering/UML consistency checks

The *Objectteering/UML* CASE tool provides over 200 consistency checks in real time, used to guarantee the quality and coherence of the model produced in the workshop. The advantages of these consistency checks are clear:

- ◆ Model consistency is checked when elements are entered, thus ensuring that inconsistent names or elements are not entered.
- ◆ Model consistency is maintained when elements are modified, thus allowing the user to avoid having to manually update all instances of the modified element.

For further information on consistency checks, please refer to the "*Consistency mechanisms*" section in chapter 1 of the *Objectteering/UML Modeler* user guide.

Removable consistency checks

However, the user may, in some modeling situations, prefer to have a certain degree of flexibility with regard to the consistency checks applied to his model. This can be the case, for example, during the preliminary phases of a project (Analysis, Specification, etc.), when users can prefer to have freer, less restrictive use of the CASE tool. Similarly, when importing models from other CASE tools which do not necessarily employ the same level of consistency checks as the *Objectteering/UML* CASE tool, it can also be useful to be able to deactivate certain checks. For this reason, certain *Objectteering/UML* consistency checks are removable. For further information on removable consistency checks, please refer to the "*Removable consistency checks*" section in chapter 3 of the *Objectteering/UML Modeler* user guide.

Chapter 5: Converting classes into EJBs and correspondence between EJB parts

Converting a class into an EJB

Converting a class into an EJB

A class can be converted into an EJB. To do this, you must create a class and label its different attributes and operations, in order to be sure what each will become once the class has been converted into an EJB. To label attributes and operations, the following EJB commands are used on attributes and operations.

Metaclass	Command applied	Use
Attribute	Set for persistence	The attribute is copied into the bean and configured to be an attribute of an EJB entity.
Attribute	Set for primary key	The attribute is copied into the bean and configured to be an attribute of an EJB entity. Furthermore, it is copied into the primary key class.
Attribute	none	Considered as being the persistent attribute of an EJB entity.
Operation	none	The operation is copied into the bean.
Operation	Set remote	The operation is copied into the bean and published in the remote class as a remote method.
Operation	Set create	The operation is copied into the bean under the name of <i>"ejbCreate"</i> and published in the home class as a create method.
Operation	Set finder	The operation is copied into the home as a finder method.

Commands used to update EJBs

The "Update the EJB with this class" command from a converted class

When a class is converted into an EJB, a link is created towards the elements of the EJB. This means that if the source class is modified and the *"Update the EJB with this class"* command run, the EJB is updated. If the functions have changed signature or their code has been modified, the corresponding functions in the bean are updated. The code previously present in these functions is overwritten.

The following window (Figure 5-1) then appears, through which the user is asked to confirm this update.

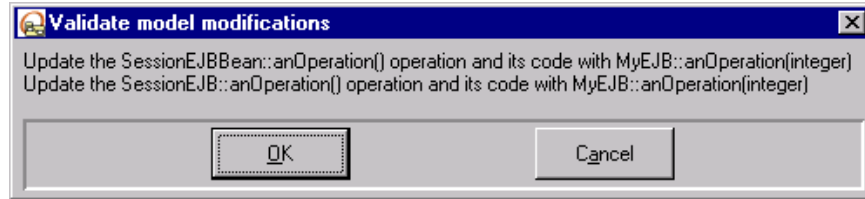


Figure 5-1. Window asking the user to confirm the update

The "Update the EJB with this class" command from one of the EJB's classes

When the *"Update the EJB with this class"* command is called from one of the EJB's classes, it propagates the differences in this part of the EJB to the other classes. As with the previous function, a window appears, in which the user is asked to confirm the transformations to be carried out.

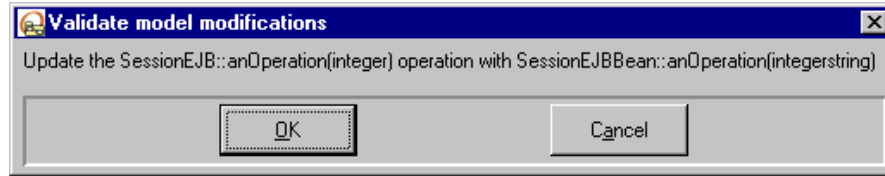


Figure 5-2. Window asking the user to confirm model modifications

This function does not impact the Java code, which remains in the bean.

Chapter 6: Objecteering/EJB functions

Package functions

Commands available on a package are shown in Figure 6-1.

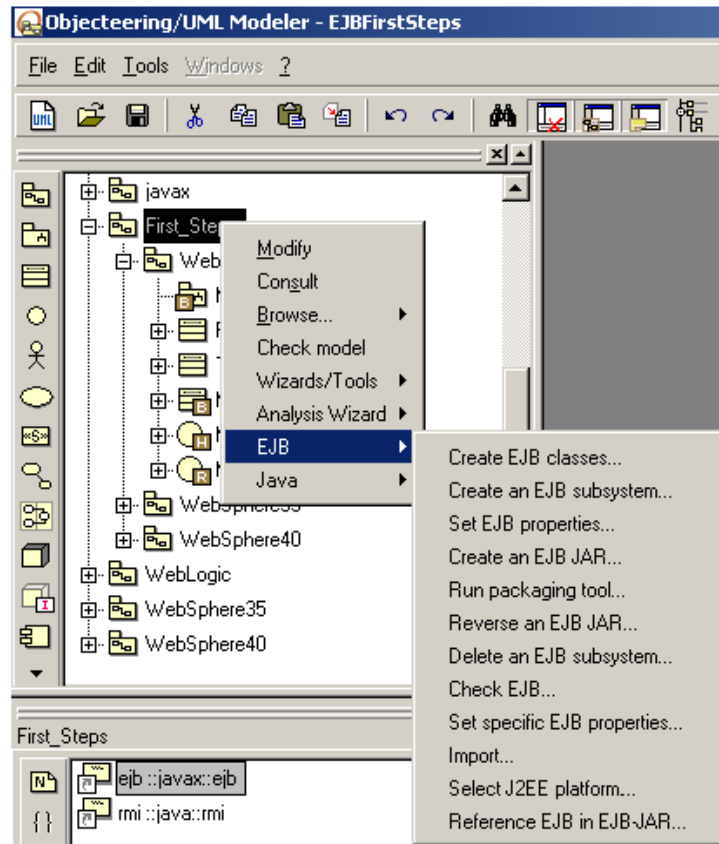


Figure 6-1. Commands available on a package

The "Create EJB Classes" command

This command creates a completely new EJB structure in the selected package.

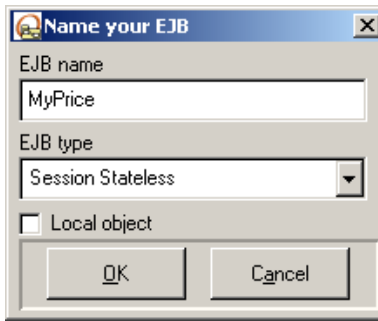


Figure 6-2. The "Name your EJB" window

Key:

- ◆ The "*EJB Name*" field is used to define the name of the EJB.
- ◆ The "*EJB Type*" field is used to define the EJB type. You can choose between "*Entity*", "*Session Stateless*" or "*Session Stateful*" or, if you are using EJB 2.0, "*Message Driven*".
- ◆ If you are using EJB 2.0, a "*Local object*" tickbox appears, giving you the possibility to create a EJB with a local and a local home instead of a remote and a home.

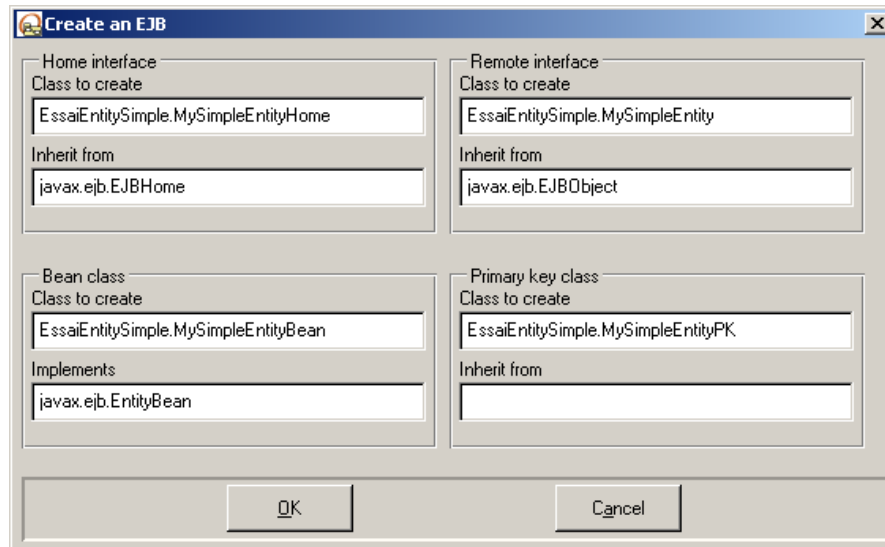


Figure 6-3. The "Create an EJB" window

Key:

- ◆ The "Class to create" fields are used to define the package and the name of the class which is to be created.
- ◆ The "Inherit from" fields are used to define the inheritance of the corresponding class.
- ◆ The "Implements" field is used to define the interface implemented by the bean class.

If you want to create an entity EJB, you should use the "Primary key class" section of the above window. You can create a primary key class for your EJB (by default, a primary key is proposed), or you can use a primary key field for your EJB. In this case, specify the type of your key, for example, "java.lang.Integer".

The "Create an EJB subsystem" command

This command creates the subsystem that contains the references on the classes of the EJB.



Figure 6-4. The "Choose a bean" window

The "*Bean class*" field defines the implementation class of the EJB.

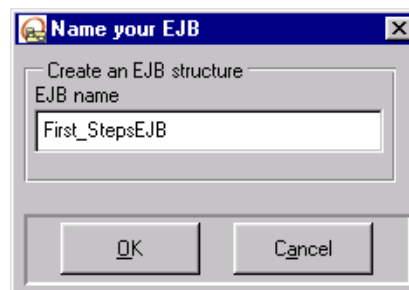


Figure 6-5. The "Name your EJB" window

The "*EJB name*" field defines the name of the EJB. The "*Set EJB properties*" window is used to define the properties of the EJB. Please refer to the "*Set EJB properties*" command for further details.

The "Reference EJB in EJB-JAR" command

This command is used to convert a package to an EJB-JAR and to reference an EJB subsystem within it. With an EJB-JAR, you can generate a JAR file containing many EJBs, each being a subsystem referenced in the EJB-JAR.

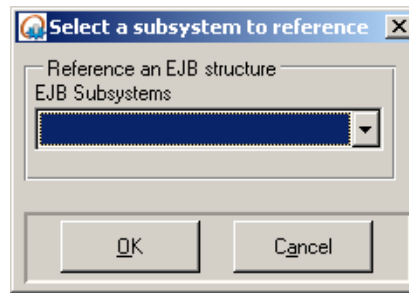


Figure 6-6. The "Select a subsystem to reference" window

To reference more than one EJB in an EJB-JAR, you must re-launch the command on the EJB-JAR.

Subsystem functions

Commands available on a subsystem are shown in Figure 6-7.

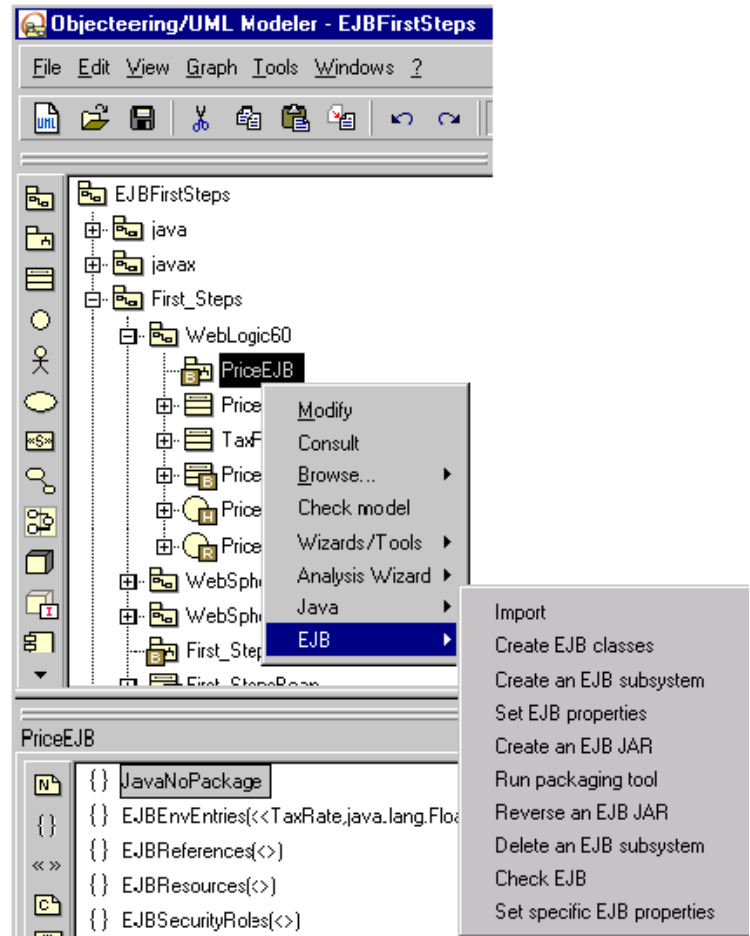


Figure 6-7. Commands available on a subsystem

The "Set EJB properties" command

This command launches a window for EJB configuration. This window is different depending on whether you are using EJB 1.1 or EJB 2.0. EJB 2.0 adds fields for "Security Identity", "Local references" and "Resource env ref."

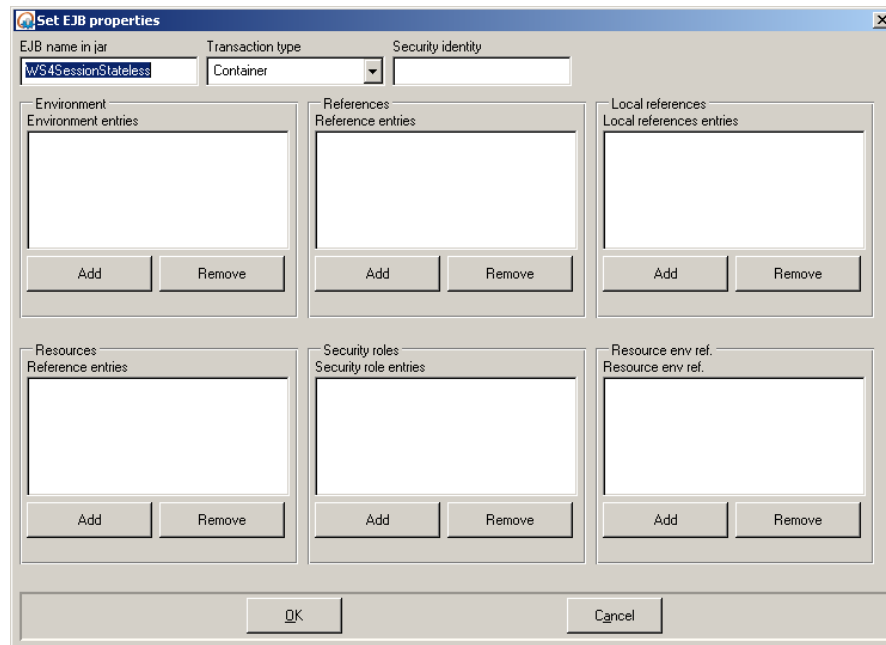


Figure 6-8. The "Set EJB properties" window

Key:

- ◆ The "EJB name in JAR" field is the name of the EJB in the JAR file.
- ◆ The "Transaction type" field is used to choose between the container transaction and the bean transaction type.
- ◆ The "Security identity" field defines the role to use for the execution of bean methods.

- ◆ "Environment entries" are variables that you wish to define in the EJB's environment. For each environment entry, you can define the name, the type and the value of the variable, as shown in Figure 6-9. These parameters are stored in the `{EJBEnvEntries}` tagged value.

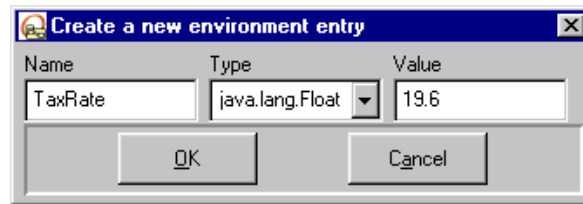


Figure 6-9. Creating a new environment entry

- ◆ The "References" field defines references to other EJBs. For each environment entry, you can define the name, the type, the home and the remote of the referenced EJB. These parameters are stored in the `{EJBReferences}` tagged value.
- ◆ The "Local references" field defines references to other local EJBs. For each local reference entry, you can define the name, the type of the EJB, its local home, its local and an optional ejb-link information. These parameters are stored in the `{EJBLocalRef}` tagged value.
- ◆ The "Resources" field is used to define EJB resources (URL, data source, etc.). For each environment entry, you can define the name, the type and the authentication of the resource. These parameters are stored in the `{EJBResources}` tagged value.
- ◆ The "Security roles" field defines the EJB's security roles. You can define the name and the link of the security role. These parameters are stored in the `{EJBSecurityRoles}` tagged value.
- ◆ The "Resource env ref" field defines references to a managed object associated with a resource. For each resource env entry, you can define the name and the type of the resource. These parameters are stored in the `{EJBResourceRef}` tagged value.
- ◆ In EJB 2.0 entity options, there is also an "Abstract schema name" field. It defines the name of the schema to be used in EJBQL queries.

The "Create an EJB JAR" command

This command is used to create a JAR file containing the files corresponding to the subsystem.

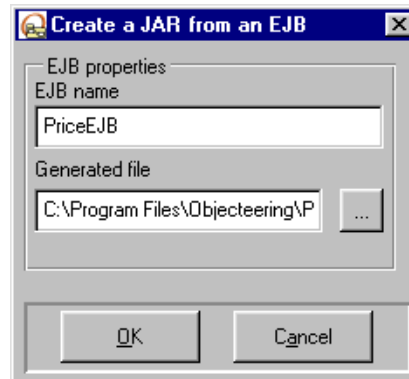


Figure 6-10. Creating a JAR from an EJB

The "Run packaging tool" command

This command runs the deployer tool of your application server. Launch this command to deploy your EJB by using the JAR file. Please refer to your application server documentation for further details.

The "Delete an EJB subsystem" command

This function will delete the selected subsystem or EJB-JAR. If you check the "*Delete recursively*" tickbox, deletion is carried out recursively on the referenced EJB subsystem and all the EJB classes.



Figure 6-11. Deleting an EJB subsystem

Subsystem and package functions

The "Check EJB" command

This function checks the validity of the selected package with regard to the EJB specifications, and can be run on both packages and subsystems.

The "Reverse an EJB JAR" command

This function allows you to create or update an EJB model from a JAR file containing one or more EJB.

If you have selected a package, this command will create a new EJB. If a subsystem or an EJB-JAR has been selected, it will update the EJB which corresponds to the subsystem.



Figure 6-12. The "Reversing an EJB JAR" window

- ◆ The "Name" field contains the name you wish to give to the EJB subsystem.
- ◆ The "JAR file to reverse" field contains the existing JAR file.

The "Select J2EE platform" command

This command opens the "*Select a platform*" window, in which you can specify which platform you are going to use and indicate its path.

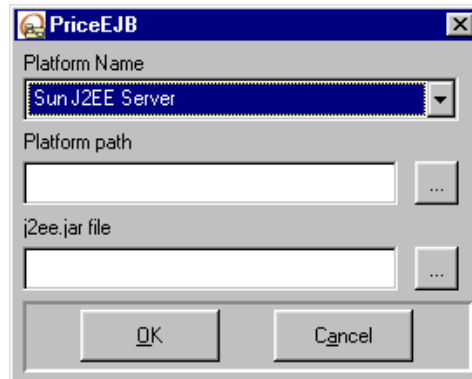


Figure 6-13. Selecting a platform

- ◆ The "*Platform name*" field is used to choose the platform you are going to use.
 - ◆ The "*Platform path*" field is used to indicate the path of the platform you have selected.
 - ◆ The "*j2ee.jar file*" field is used to indicate the location of the J2EE .jar file.
-

Class functions

Commands available on a class are shown in Figure 6-14.

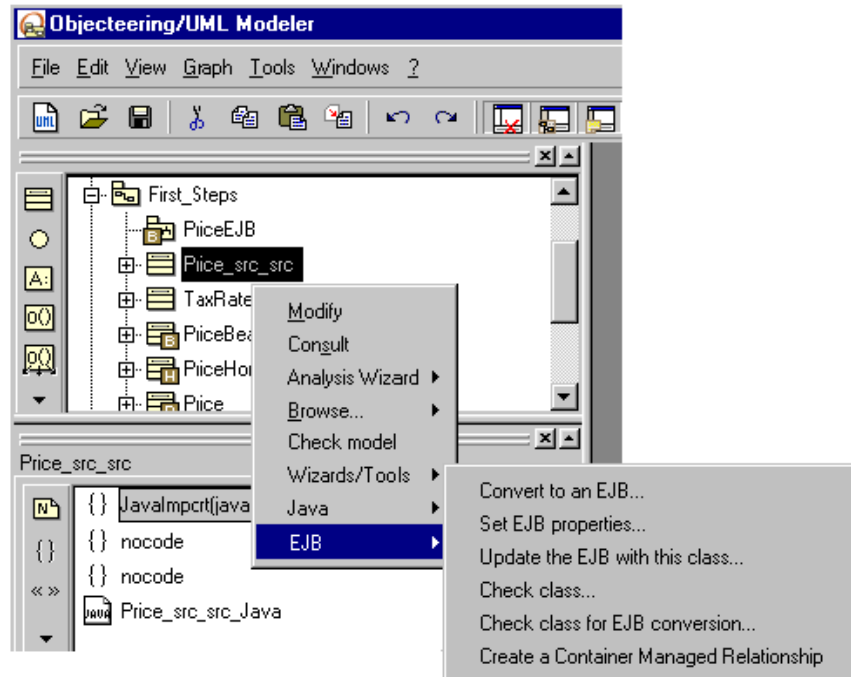


Figure 6-14. Commands available on a class

The "Convert to an EJB" command

This function creates a new EJB from an existing class. For further details, please refer to the "*Create EJB classes*" command detailed in the "*Package functions*" section of the current chapter of this user guide.

The "Set EJB properties" command

This function allows you to modify the session type property of a session EJB.

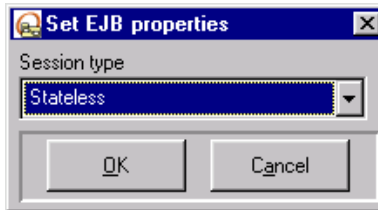


Figure 6-15. Setting EJB properties

This property will be stored in the `{EJBSessionType}` tagged value on the home class.

The "Update the EJB with this class" command

This function updates the EJB classes with regard to the selected class.

Note: The *EJBFinderMethods* are not updated, because the transfer of the finders depends on EJB configuration (Container Managed Persistence or Bean Managed Persistence), which is not at class model level.

The "Check class" command

This function checks the validity of the selected EJB class with regard to EJB specifications.

The "Check class for EJB conversion" command

This function checks the validity of the selected class which is to be converted with regard to the EJB specifications.

The "Create a container managed relationship" command

This function is used to create a container managed relationship between two entity EJBs. This command must be launched on the bean class of an EJB entity.

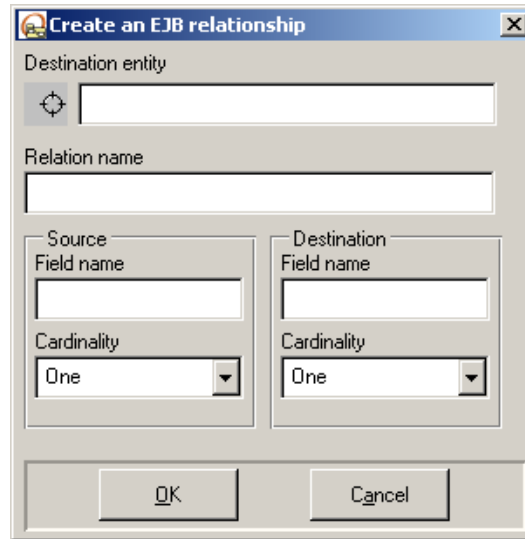


Figure 6-16. Creating an EJB relationship

- ◆ The "*Destination entity*" field is a drop box. You must drag into this field the other bean to be linked to the CMR you are creating.
- ◆ The "*Relation name*" field is the name of the relation to be created.
- ◆ The "*Source*" and "*Destination*" zones concern AssociationEnds connected to this association.
- ◆ The "*Field name*" field is used to specify the name of the AssociationEnd.
- ◆ The "*Cardinality*" is used to specify the cardinality of this role (One or Many).

All characteristics of the container managed relation are on the UML Association. The `<cascade-delete />` is generated if a role has an *Association Kind* of *Composition*. An `AssociationEnd` must be declared as abstract, and have as its type *Collection* or *Set*. The only accessors needed are *get_all* and *set_all*, which means that the `AssociationEnd` must have a `{JavaFilterAccessor}` tagged value with the *card*, *erase_by_element* and *append* values.

Operation functions

Commands available on an operation are shown in Figure 6-17.

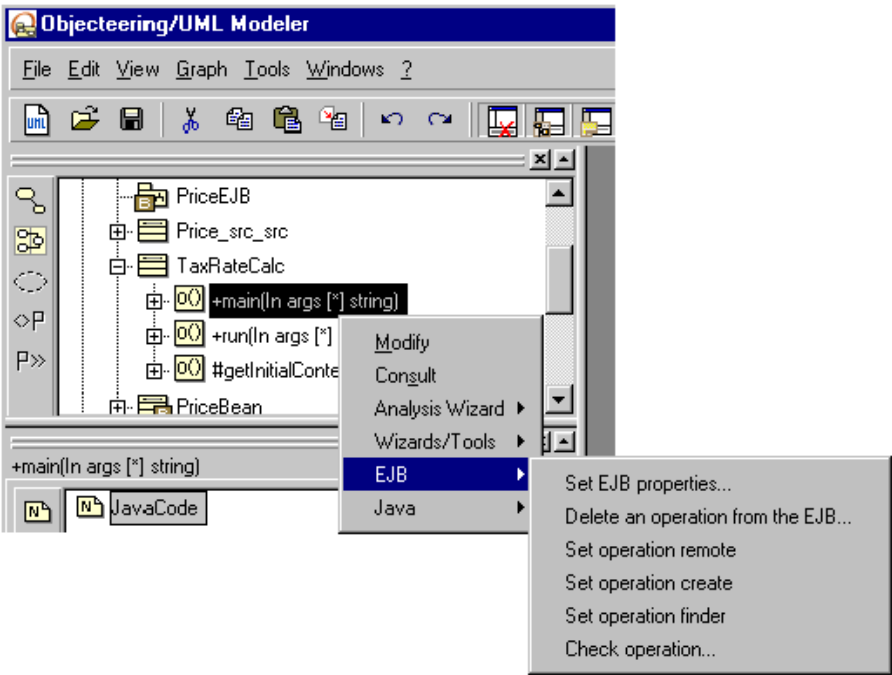


Figure 6-17. Commands available on an operation

The "Set EJB properties" command

This function sets the transaction properties and security roles of the selected operation.

If you are using EJB 2.0 and you launch it on a finder operation other than *findByPrimaryKey*, there are two additional options, used to define result type mapping and the EJBQL query corresponding to the finder.

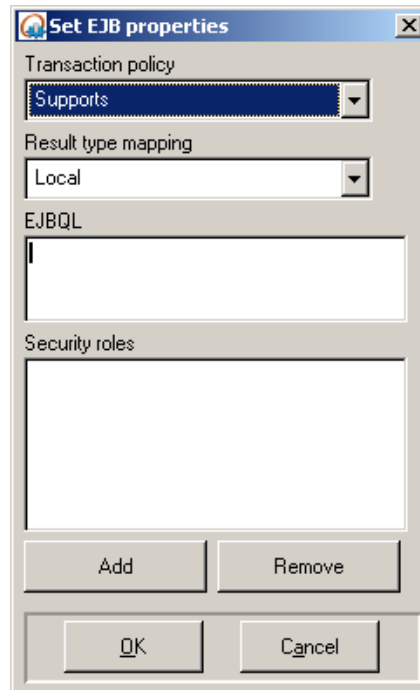


Figure 6-18. Setting EJB properties

The "Delete an operation from the EJB" operation

This command deletes the selected operation.

The "Set operation remote" command

This function sets the remote attributes for the selected operation.

The "Set operation create" command

This command sets the "*create*" operation attributes on the selected operation.

The "Set operation finder" command

This function sets the "*finder*" operation attributes on the selected operation.

The "Check operation" command

This command checks the validity of the operation with regard to the EJB specifications.

Tagged value types

Overview

The tagged values provided by Objecteering/UML allow you to adapt EJB semantics to a UML model, in order to correctly generate all the EJB notions.

UML extensions for EJB 1.1 are standardized by the "*UML Profile for EJB*", defined by the JCP (JSR 26). Currently, there is no standard for EJB 2.0 modeling. UML extensions specific to EJB 2.0 are not standardized by the JCP.

Tagged values on a home interface class

The ... tagged value	with parameters ...	EJB version	is used to ...
{EJBSessionType}	"Stateless" or "Stateful"	1.1 and 2.0	determine whether or not the EJB should keep its state.

Tagged values on an operation

The ... tagged value	with parameters ...	EJB version	is used to ...
{EJBRoleName}	List of strings separated by commas	1.1 and 2.0	indicate the EJB security role which can call the operation.
{EJBTransAttribute}	"Not Supported", "Supports", "Required", "RequireNew", "Mandatory" or "Never"	1.1 and 2.0	define the transaction policy of the operation.
{EJBUncallable}		2.0	define the operation as being uncalleable.
{EJBResultTypeMapping}	"Local" or "Remote"	2.0	determine the result type mapping, on an operation stereotyped EJBFinderMethod.

Tagged values on a subsystem stereotyped «EJBEnterpriseBean»

The ... tagged value	with ... parameters	EJB version	is used to ...
{EJBEnvEntries}	List of elements <name, type, value>	1.1 and 2.0	define the environment variables used by the EJB.
{EJBNameInJar}	A string	1.1 and 2.0	define the name of the EJB in the ejb-jar.xml file.
{EJBReferences}	List of elements <name, type, home, remote>	1.1 and 2.0	define the EJB referenced by the current EJB.
{EJBResources}	List of elements <name, type, auth>	1.1 and 2.0	define the resource factories used by the EJB.
{EJBSecurityRole}	List of elements <name, link>	1.1 and 2.0	define the name security roles than can call the functions of the EJB.
{EJBTransType}	"Bean" or "Container"	1.1 and 2.0	define the type of persistence.
{EJBReentrant}	"True" or "False"	1.1 and 2.0	define if the EJB is re-incoming.
{EJBPersistenceType}	"Bean" or "Container"	1.1 and 2.0	define the persistence type of an EJB Entity.
{EJBLocalRef}	List of elements <name, type, local-home, local, link>	2.0	define a local EJB referenced by this EJB.
{EJBResourceRef}	List of elements <name, type>	2.0	define a reference to a managed object associated with a resource.

The ... tagged value	with parameters ...	EJB version	is used to ...
{EJBSecurityIdentity}	A string	2.0	define the role to use for the execution of the bean methods.
{EJBAbstractSchemaName}	A string	2.0	define the name of the schema to use in EJBQL queries.
{EJBMessageSelector}	A String	2.0	define a condition of message selection on a Message Driven EJB queue.
{EJBAcknowledgeMode}	"Auto-acknowledge" or "Dups-ok-acknowledge"	2.0	define the acknowledge mode for an EJB message driven.
{EJBMessageDrivenDestination}	A couple of value, <type, durability> where type is "javax.jms.Queue" or "javax.jms.Topic", and durability is "Durable" or "NonDurable".	2.0	define the message queue to use with an EJB Message Driven, and options for its subscription.

Note types

The command ...	EJB version	is used to ...
EJBQL	2.0	contain the implementation of a finder written in EJBQL, when placed on a finder operation.

Stereotypes

Stereotypes on an operation

The ... stereotype	EJB version	is used to ...
<<EJBCreateMethod>>	1.1 and 2.0	define a creation method for home interfaces. This is a specialization of the <i>"EJBHomeMethod"</i> .
<<EJBFinderMethod>>	1.1 and 2.0	define a search method for the home interface. This is a specialization of the <i>"EJBHomeMethod"</i> .
<<EJBRemoteMethod>>	1.1 and 2.0	define a method of the remote interface.
<<EJBOnMessage>>	2.0	define an operation to be the <i>"onMessage"</i> operation of an EJB message driven.

Stereotypes on a class

The ... stereotype	EJB version	is used to ...
<<EJBRemoteInterface>>	1.1 and 2.0	define the remote interface of the EJB.
<<EJBSessionHomeInterface>>	1.1 and 2.0	define the home interface of a session EJB.
<<EJBEntityHomeInterface>>	1.1 and 2.0	define the home interface of an entity EJB.
<<EJBImplementation>>	1.1 and 2.0	define the implementation class of an EJB.
<<EJBObjectInterface>>	2.0	define the local interface of an EJB.
<<EJBObjectSessionHomeInterface>>	2.0	define the local home interface of a session EJB.
<<EJBObjectEntityHomeInterface>>	2.0	define the local home interface of an entity EJB.

Stereotypes on a link

The ... stereotype	EJB version	is used to ...
<<EJBPrimaryKey>>	1.1 and 2.0	specify that the destination of the link is the primary key class of the EJB entity.
<<EJBRealizeHome>>	1.1 and 2.0	define a link between the home interface and the implementation bean.
<< EJBRealizeRemote>>	1.1 and 2.0	define a link between the remote interface and the implementation bean.
<<EJBClientJar>>	1.1 and 2.0	specify the ejb-client-jar relation between two ejb-jar files.

Stereotypes on a subsystem

The ... stereotype	EJB version	is used to ...
<<EJBSessionBean>>	1.1 and 2.0	define an EJB session.
<<EJBEntityBean>>	1.1 and 2.0	define an EJB entity.
<<EJBMessageDrivenBean>>	2.0	define an EJB message driven.

Stereotypes on a package

The ... stereotype	EJB version	is used to ...
<<JavaArchiveFile>>	1.1 and 2.0	indicate that the package represents a JAR file.
<<EJB-JAR>>	1.1 and 2.0	indicate that the package represents an EJB-JAR file which contains several EJBs.

Stereotypes on a component

The ... stereotype	EJB version	is used to ...
<<EJBDescriptor>>	1.1 and 2.0	indicate the XML deployment file.
<<JavaClassFile>>	1.1 and 2.0	indicate that the package represents a Java class file.

Stereotypes on an association

The ... stereotype	EJB version	is used to ...
<<EJBRelationship>>	2.0	declare, between two <<EJBImplementations>>, a container managed relationship between two EJB entities.

Stereotypes on an attribute

The ... stereotype	EJB version	is used to ...
<<EJB_CMP_Field>>	1.1 and 2.0	define an attribute of an EJBImplementation to be saved by container managed persistence.
<<EJBPrimaryKeyField>>	1.1 and 2.0	define an attribute of an EJBImplementation to be a key for container managed persistence.

Chapter 7: Parameterization

Defining module parameters

Introduction

The following elements can be parameterized by the user:


- ◆ Application Server options
- ◆ Default Inheritance
- ◆ Reverse options
- ◆ Validation options
- ◆ Interface options
- ◆ IONA iPortal Application Server
- ◆ External edition options

For further information, please refer to the "*Parameter sets*" section in the current chapter of this user guide.

Parameter sets

Configuring the Objecteering/EJB module

The *Objecteering/EJB* module can be parameterized through the "*Modifying configuration*" dialog box (as shown in Figure 7-1 below), which is launched by

clicking on the  "*Modify module parameter configuration*" icon.

External edition parameters

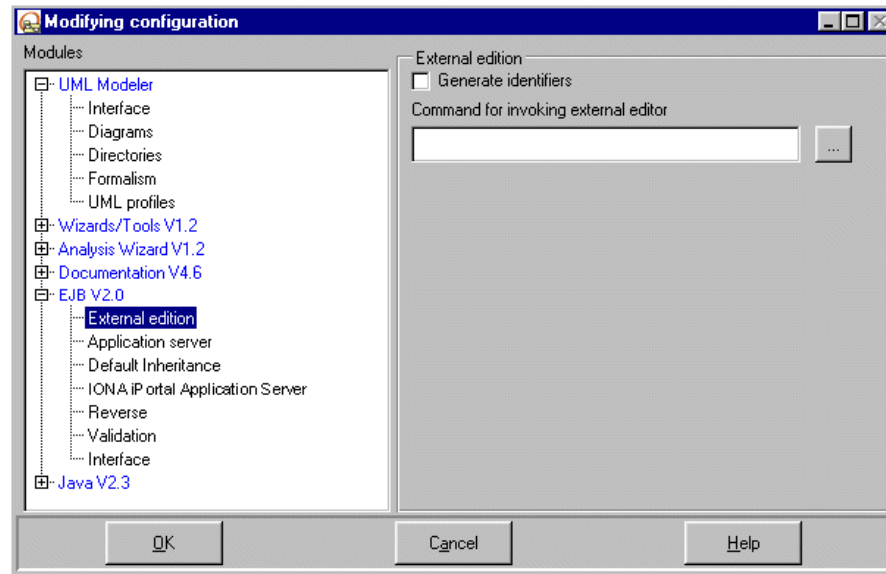


Figure 7-1. The "External edition" sub-category of parameters

Key:

- ◆ The "*Generate identifiers*" tickbox is used to indicate whether or not markers used to retrieve text entered using an external text editor should be generated.
- ◆ The "*Command for invoking external editor*" field is used to specify the name of the editor used to edit the files generated in Objectteering/UML.

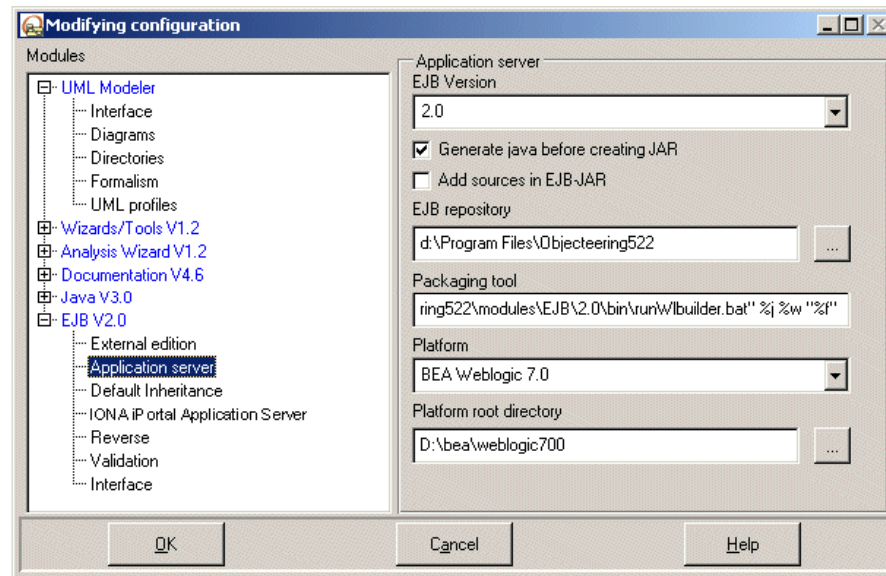
Application server parameters

Figure 7-2. The "Application server" sub-category of parameters

Key:

- ◆ The "*EJB Version*" field is used to indicate the EJB version to generate (either 1.1 or 2.0).
- ◆ The "*Generate java before creating JAR*" tickbox should be checked if you wish to automatically generate Java code before creating the JAR file.
- ◆ The "*Add sources in EJB JAR*" tickbox should be checked if you wish to include the source files in the EJB JAR file.
- ◆ The "*EJB repository*" field is used to define the path where you wish to store your EJB.
- ◆ The "*Packaging tool*" field is used to define the command used to launch the descriptor editor of your application server. There exists a substitution system for this command line:
 - ◆ %f : JAR file generated by Objectteering/UML
 - ◆ %b : JAR file with the -built suffix
 - ◆ %w : platform root directory
 - ◆ %d : platform root drive
 - ◆ %j : JDK path
- ◆ The "*Platform*" field is used to select your application server.
- ◆ The "*Platform root directory*" is used to indicate the root directory of your application server.

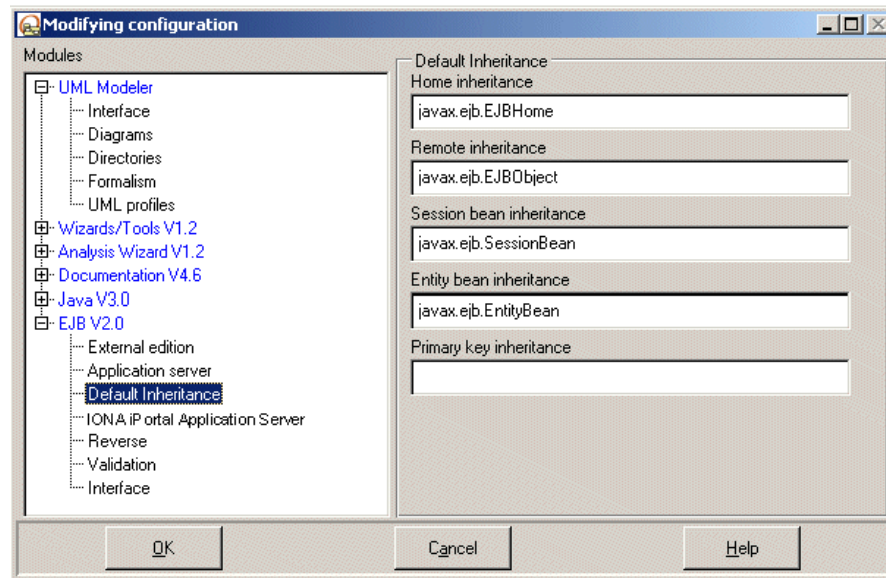
Default inheritance parameters

Figure 7-3. The "Default inheritance" sub-category of parameters

In this sub-category, you can enter the default inheritance of each part of the EJB.

IONA iPortal Application Server parameters

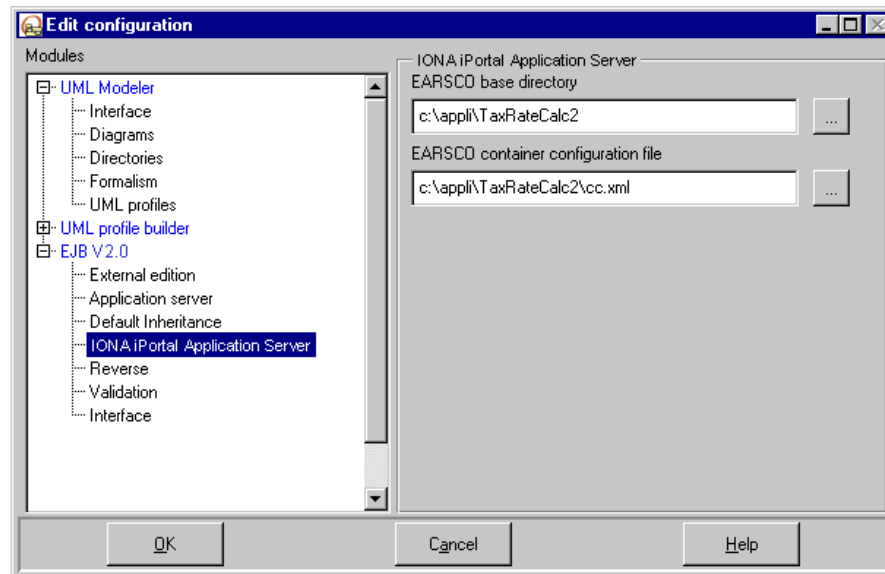


Figure 7-4. The "IONA iPortal Application Server" sub-category of parameters

Key:

- ◆ "EARSCO base directory": This is where the EARSCO framework of your application is created.
- ◆ "EARSCO container configuration file": This field is used to indicate the EARSCO container configuration file, which is the XML file containing deployment parameters.

Reverse parameters

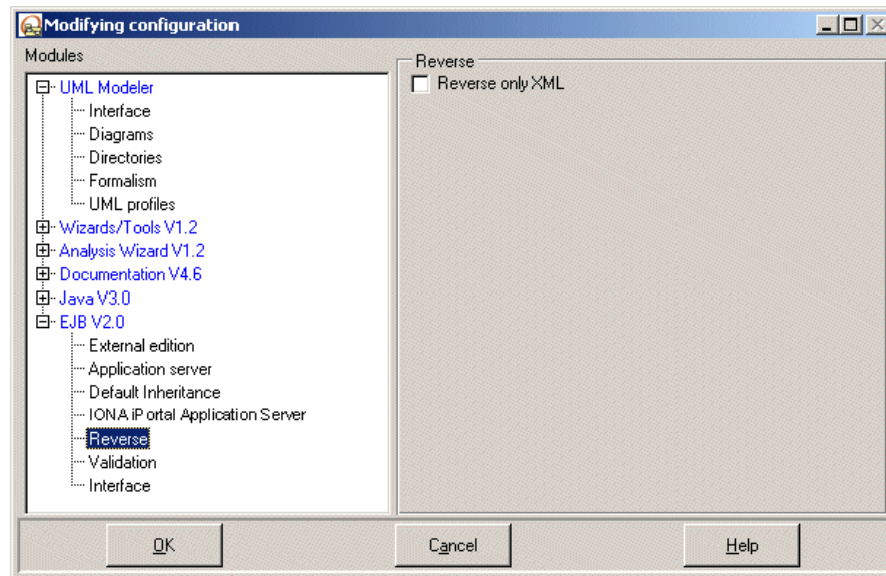


Figure 7-5. The "Reverse" sub-category of parameters

Check the "Reverse only XML" tickbox to reverse only the XML file of your EJB.

Validation parameters

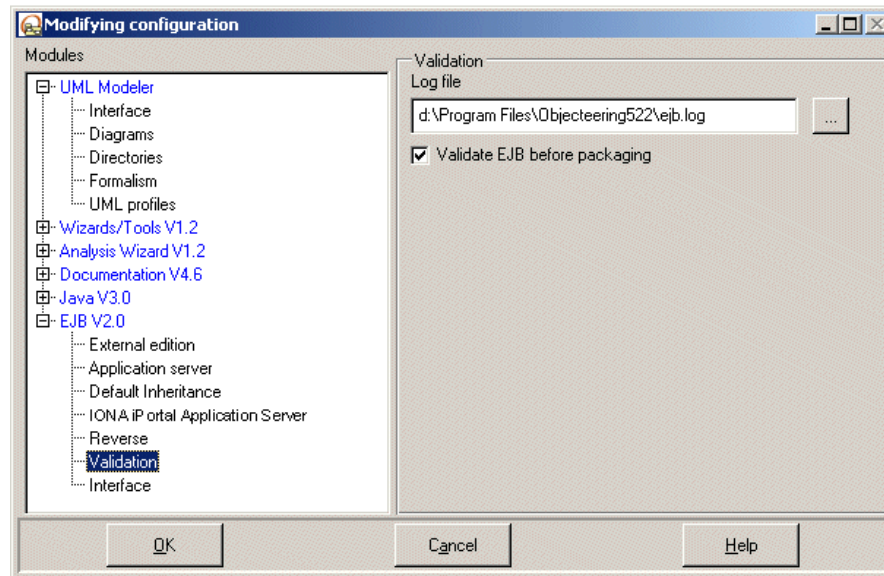


Figure 7-6. The "Validation" sub-category of parameters

In the "Log file" field, define the validation log file.

Use the "Validate EJB before packaging" tickbox to validate your modeling before creating your JAR file.

Interface parameters

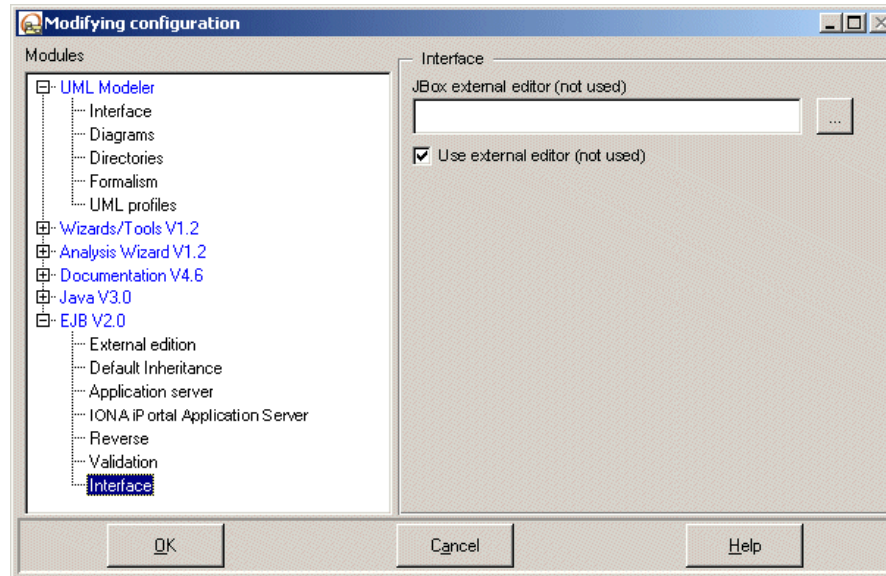


Figure 7-7. The "Interface" sub-category of parameters

These module parameters are not currently implemented.

Chapter 8: Specific EJB properties

WebLogic 7.0 and Objectteering/UML

Weblogic 7.0 can be used with Objectteering/UML to deploy your EJBs.

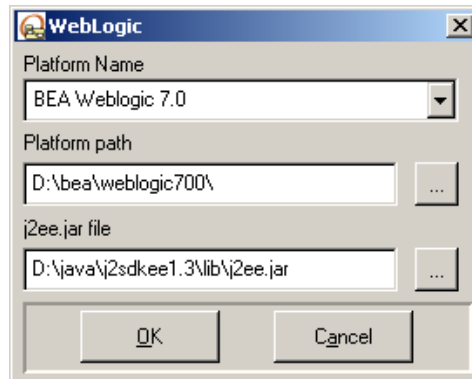


Figure 8-1. Selecting WebLogic 7.0

In the window shown in Figure 8-1, select BEA WebLogic 7.0 as the platform name, and WebLogic's installation path as the platform path.

Using the "*Run packaging tool*" command, you launch WebLogic builder on the JAR you created in Objectteering/UML. In this tool, EJB properties can be modified. When you close this tool, the JAR is analyzed and modifications transferred to the model. Modifications to EJB properties are transferred to the model. Modifications to WebLogic-specific properties are transferred into specificXml notes on the subsystem. This means that if you re-generate the JAR file, this information is added to the file, so as to avoid losing your modifications.

WebLogic 6.0 and Objectteering/UML

Some specific information is needed by WebLogic 6.0 in its deployment descriptor. This information is defined using the "*Set specific EJB properties*" command of the "*EJB*" menu.

Details on the mapping between the dialog boxes and the XML files are given in the following pages.

The main dialog box is shown below.

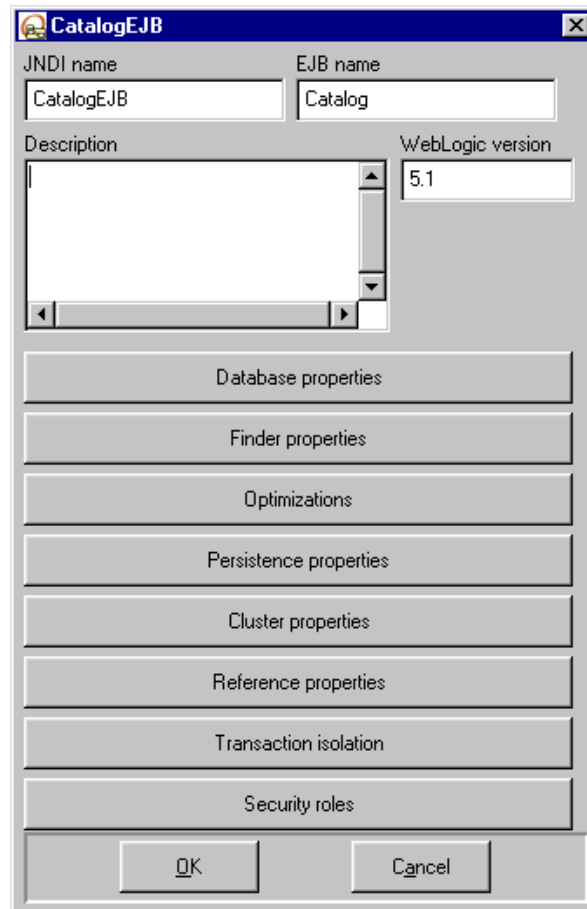


Figure 8-2. The main dialog box

The ... field	corresponds to ... of the weblogic-ejb-jar file
JNDI name	weblogic-ejb-jar.weblogic-enterprise-bean.jndi-name
EJB name	weblogic-ejb-jar.weblogic-enterprise-bean.ejb-name
Description	weblogic-ejb-jar.description
WebLogic version	weblogic-ejb-jar.weblogic-version

The database properties dialog box is shown in Figure 8-3 below.



Figure 8-3. The database properties dialog box

The ... field	corresponds to ... of the weblogic-rdbms-persistence file
Data source name	weblogic-rdbms-bean.pool-name
Schema name	weblogic-rdbms-bean.schema-name
Table name	weblogic-rdbms-bean.table-name
Attribute map	weblogic-rdbms-bean.attribute-map
Attribute name	weblogic-rdbms-bean.object-link.bean-field
Column name	weblogic-rdbms-bean.object-link.dbms-column
Isolation level	weblogic-rdbms-bean.options.transaction-isolation

The finder properties dialog box is shown in Figure 8-4 below.

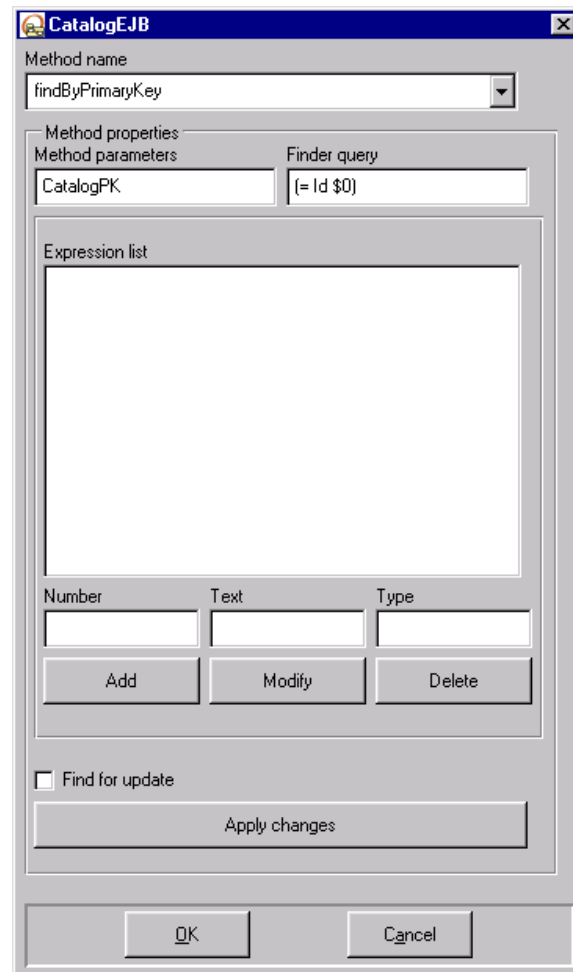
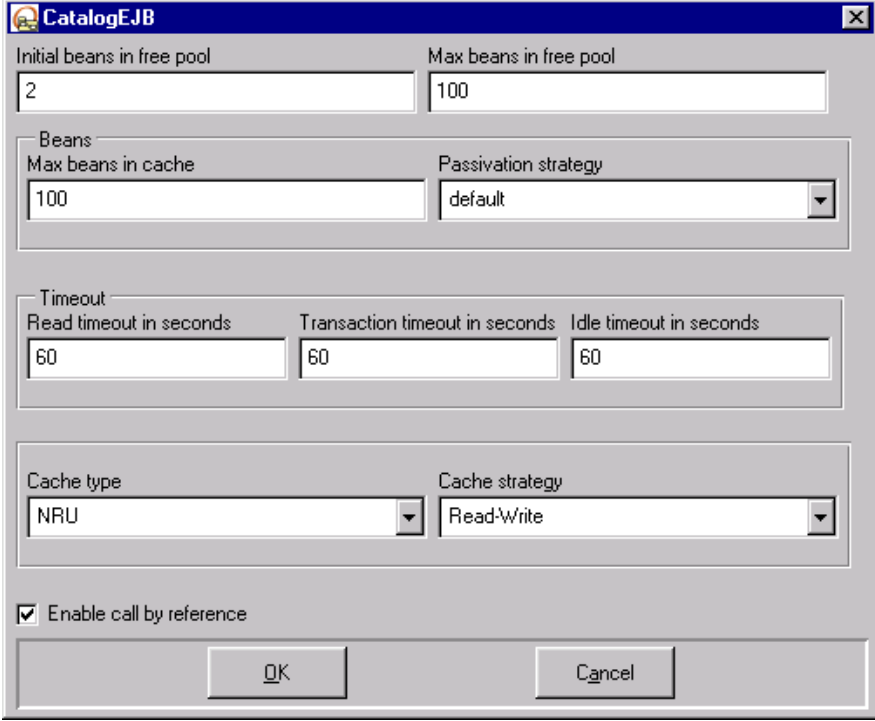


Figure 8-4. The finder properties dialog box

The ... field	corresponds to ... of the weblogic-rdbms-persistence file
Method name	weblogic-rdbms-bean.finder-list.finder.method-name
Method parameters	weblogic-rdbms-bean.finder-list.finder.method-params.method-param
Expression list	weblogic-rdbms-bean.finder-list.finder.finder-expression
Number	weblogic-rdbms-bean.finder-list.finder.finder-expression.expression-number
Text	weblogic-rdbms-bean.finder-list.finder.finder-expression.expression-text
Type	weblogic-rdbms-bean.finder-list.finder.finder-expression.expression-type
Find for update	weblogic-rdbms-bean.finder-list.finder.finder-options.find-for-update

The optimization dialog box is shown in Figure 8-5 below.



The image shows a Java Swing dialog box titled "CatalogEJB". It contains several groups of settings for EJB optimization. The first group has two text fields: "Initial beans in free pool" with the value "2" and "Max beans in free pool" with the value "100". The second group, labeled "Beans", has "Max beans in cache" set to "100" and a "Passivation strategy" dropdown menu set to "default". The third group, labeled "Timeout", has three text fields: "Read timeout in seconds" (60), "Transaction timeout in seconds" (60), and "Idle timeout in seconds" (60). The fourth group has two dropdown menus: "Cache type" set to "NRU" and "Cache strategy" set to "Read-Write". At the bottom, there is a checked checkbox labeled "Enable call by reference". The dialog box has "OK" and "Cancel" buttons at the bottom right.

Property	Value
Initial beans in free pool	2
Max beans in free pool	100
Max beans in cache	100
Passivation strategy	default
Read timeout in seconds	60
Transaction timeout in seconds	60
Idle timeout in seconds	60
Cache type	NRU
Cache strategy	Read-Write
Enable call by reference	<input checked="" type="checkbox"/>

Figure 8-5. The optimization dialog box

The ... field	corresponds to ... of the weblogic-ejb-jar file
Initial beans in free pool	weblogic-ejb-jar.weblogic-enterprise-bean.caching-descriptor.initial-beans-in-free-pool
Max beans in free pool	weblogic-ejb-jar.weblogic-enterprise-bean.caching-descriptor.max-beans-in-free-pool
Max beans in cache	weblogic-ejb-jar.weblogic-enterprise-bean.caching-descriptor.max-beans-in-cache
Passivation strategy	weblogic-ejb-jar.weblogic-enterprise-bean.caching-descriptor.passivation-strategy
Read timeout in seconds	weblogic-ejb-jar.weblogic-enterprise-bean.caching-descriptor.read-timeout-seconds
Transaction timeout in seconds	weblogic-ejb-jar.weblogic-enterprise-bean.transaction-descriptor.trans-timeout-seconds
Idle timeout in seconds	weblogic-ejb-jar.weblogic-enterprise-bean.caching-descriptor.idle-timeout-seconds
Cache type	weblogic-ejb-jar.weblogic-enterprise-bean.caching-descriptor.cache-type
Cache strategy	weblogic-ejb-jar.weblogic-enterprise-bean.caching-descriptor.cache-strategy
Enable call by reference	weblogic-ejb-jar.weblogic-enterprise-bean.enable-call-by-reference

The persistence properties dialog box is shown in Figure 8-6 below.

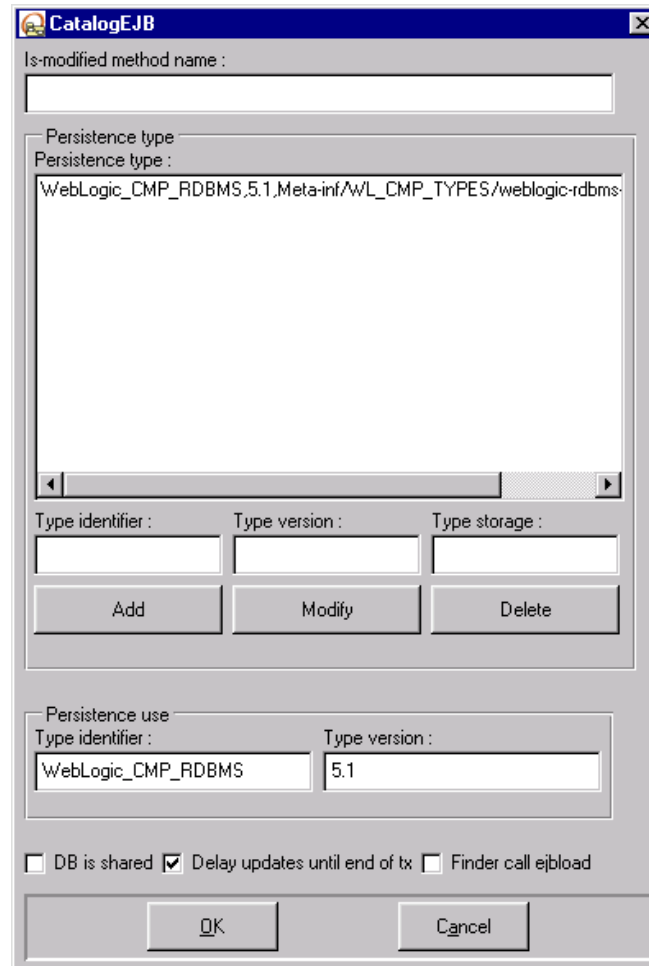


Figure 8-6. The persistence properties dialog box

The ... field	corresponds to ... of the weblogic-ejb-jar file
Is-modified method name	weblogic-ejb-jar.weblogic-enterprise-bean.persistence-descriptor.is-modified-method-name
Persistence type	weblogic-ejb-jar.weblogic-enterprise-bean.persistence-descriptor.persistence-type
Type identifier in Persistence type	weblogic-ejb-jar.weblogic-enterprise-bean.persistence-descriptor.persistence-type.type-identifier
Type version in Persistence type	weblogic-ejb-jar.weblogic-enterprise-bean.persistence-descriptor.persistence-type.type-version
Type storage in Persistence type	weblogic-ejb-jar.weblogic-enterprise-bean.persistence-descriptor.persistence-type.type-storage
Type identifier in Persistence use	weblogic-ejb-jar.weblogic-enterprise-bean.persistence-descriptor.persistence-use.type-identifier
Type version in Persistence use	weblogic-ejb-jar.weblogic-enterprise-bean.persistence-descriptor.persistence-use.type-version
DB is shared	weblogic-ejb-jar.weblogic-enterprise-bean.persistence-descriptor.db-is-shared
Delay updates until end of tx	weblogic-ejb-jar.weblogic-enterprise-bean.persistence-descriptor.delay-updates-until-end-of-tx
Finder call ejbLoad	weblogic-ejb-jar.weblogic-enterprise-bean.persistence-descriptor.finders-call-ejbload
Stateful session persistent store directory	weblogic-ejb-jar.weblogic-enterprise-bean.persistence-descriptor.stateful-session-persistent-store-dir

The clusters properties dialog box is shown in Figure 8-7 below.

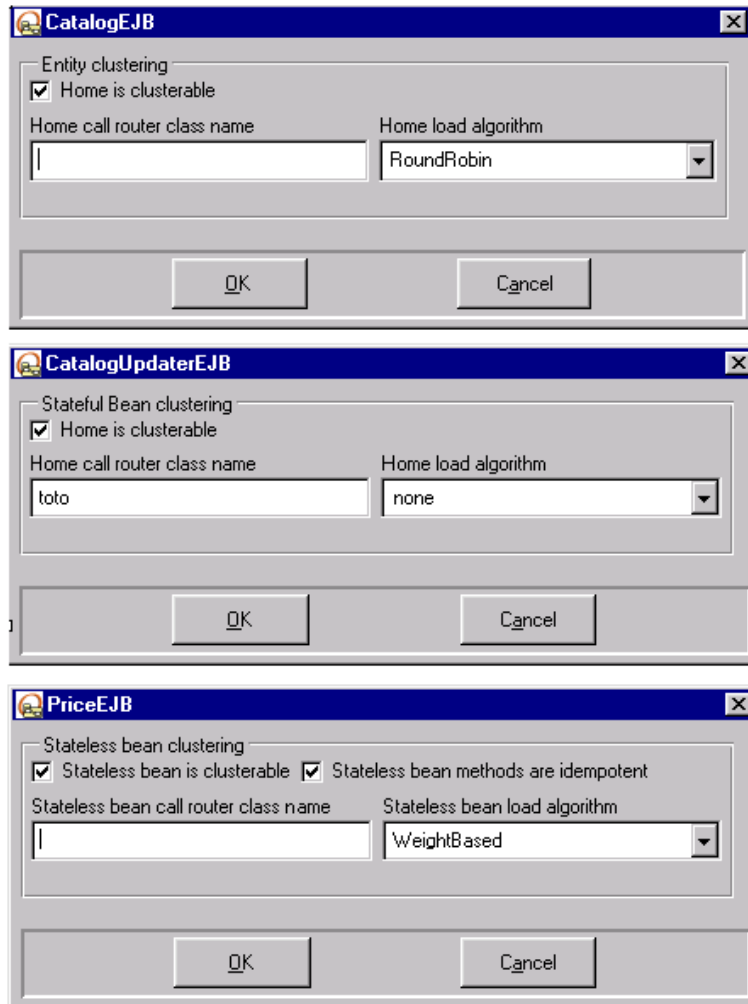


Figure 8-7. The clusters properties dialog box

The ... field	corresponds to ... of the weblogic-ejb-jar file
Home is clusterable	weblogic-ejb-jar.weblogic-enterprise-bean.clustering-descriptor.home-is-clusterable
Home call router class name	weblogic-ejb-jar.weblogic-enterprise-bean.clustering-descriptor.home-call-router-class-name
Home load algorithm	weblogic-ejb-jar.weblogic-enterprise-bean.clustering-descriptor.home-load-algorithm
Stateless bean is clusterable	weblogic-ejb-jar.weblogic-enterprise-bean.clustering-descriptor.stateless-bean-is-clusterable
Stateless bean methods are idempotent	weblogic-ejb-jar.weblogic-enterprise-bean.clustering-descriptor.stateless-bean-methods-are-idempotent
Stateless bean call router class name	weblogic-ejb-jar.weblogic-enterprise-bean.clustering-descriptor.stateless-bean-call-router-class-name
Stateless bean load algorithm	weblogic-ejb-jar.weblogic-enterprise-bean.clustering-descriptor.stateless-bean-load-algorithm

The references properties dialog box is shown in Figure 8-8 below.

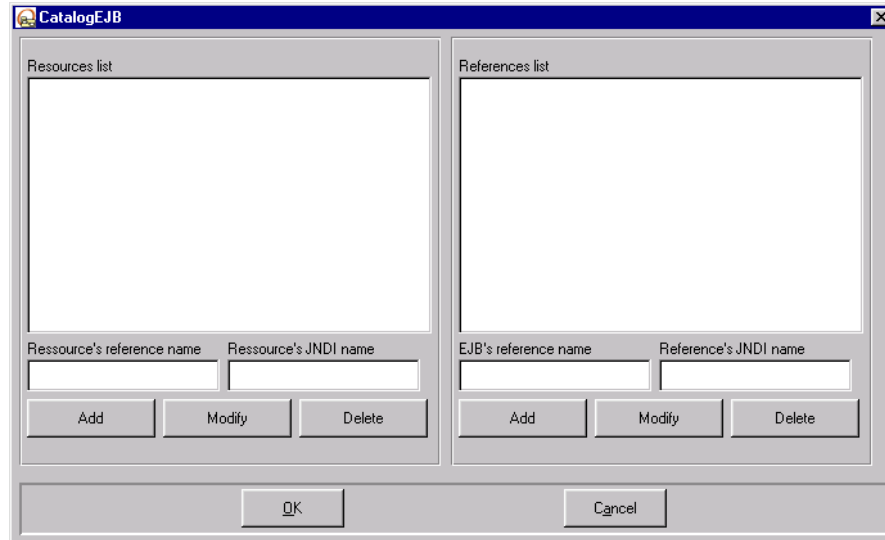


Figure 8-8. The references properties dialog box

The ... field	corresponds to ... of the weblogic-ejb-jar file
Resources list	weblogic-ejb-jar.weblogic-enterprise-bean.reference-descriptor.resource-description
Resource's reference name	weblogic-ejb-jar.weblogic-enterprise-bean.reference-descriptor.resource-description.res-ref-name
Resource's JNDI name	weblogic-ejb-jar.weblogic-enterprise-bean.reference-descriptor.resource-description.jndi-name
References list	weblogic-ejb-jar.weblogic-enterprise-bean.reference-descriptor.ejb-reference-description
EJB's reference name	weblogic-ejb-jar.weblogic-enterprise-bean.reference-descriptor.ejb-reference-description.ejb-ref-name
Reference's JNDI name	weblogic-ejb-jar.weblogic-enterprise-bean.reference-descriptor.ejb-reference-description.jndi-name

The transaction isolation dialog box is shown in Figure 8-9 below.

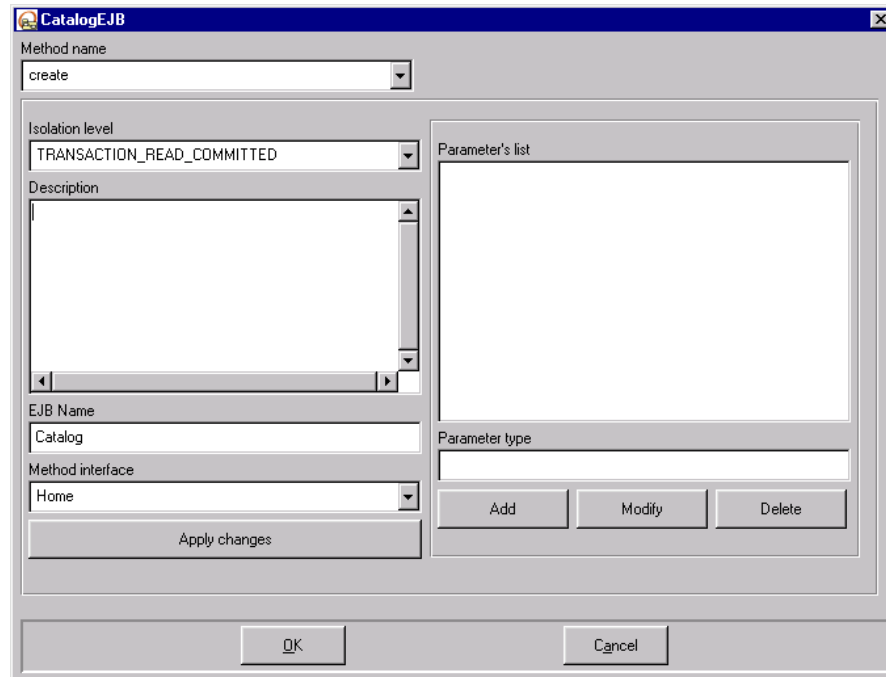


Figure 8-9. The transaction isolation dialog box

The ... field	corresponds to ... of the weblogic-ejb-jar file
Method name	weblogic-ejb-jar.weblogic-enterprise-bean.transaction-isolation.method.method-name
Isolation level	weblogic-ejb-jar.weblogic-enterprise-bean.transaction-isolation.isolation-level
Description	weblogic-ejb-jar.weblogic-enterprise-bean.transaction-isolation.method.description
EJB Name	weblogic-ejb-jar.weblogic-enterprise-bean.transaction-isolation.method.ejb-name
Method interface	weblogic-ejb-jar.weblogic-enterprise-bean.transaction-isolation.method.method-intf
Parameter's list	weblogic-ejb-jar.weblogic-enterprise-bean.transaction-isolation.method.method-params
Parameter type	weblogic-ejb-jar.weblogic-enterprise-bean.transaction-isolation.method.method-params.method-param

The security roles dialog box is shown in Figure 8-10 below.

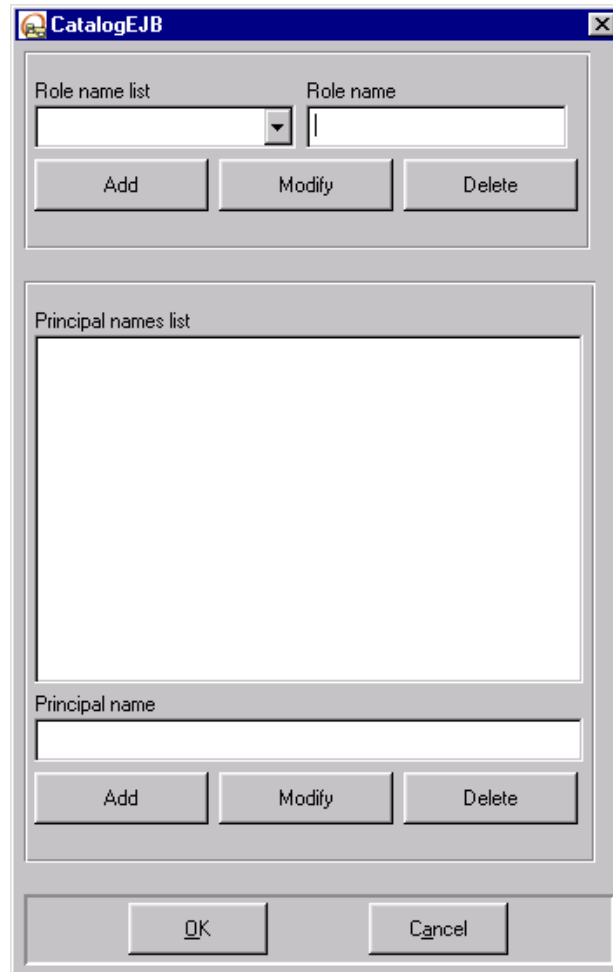


Figure 8-10. The security roles dialog box

The ... field	corresponds to ... of the weblogic-ejb-jar file
Role name list	weblogic-ejb-jar.security-role-assignment.role-name
Principal name list	weblogic-ejb-jar.security-role-assignment.principal-name

Note 1: In the "*Finders properties*" and the "*Transaction isolation*" dialog boxes, you must click on the "*Apply*" button to save changes.

Note 2: Only EJB 1.1 is supported by the *Objectteering/EJB* module.

You can deploy your EJB by carrying out the following steps:

- 1 - Configure your environment using the "setenv" command of the weblogic_installation_path/wlserver6.0sp1/config/system directory.
- 2 - Use the java weblogic.ejbc source_file dest_file command to create your EJB's container.

For more details on how to deploy the EJB on the WebLogic server, please refer to the WebLogic documentation.

WebSphere 3.5 and Objectteering/UML

After you have created the jar file with Objectteering/UML, follow these steps:

- 1 - Select the subsystem of your EJB and run the "*Run packaging tool*" command from the EJB menu.
- 2 - This command will run "*jetace*" from IBM. This tool will help you define options for your EJB's container.
- 3 - Use the WebSphere console to deploy your EJB on the server.

Please refer to the WebSphere documentation for more details.

WebSphere 4.0 and Objecteering/UML

Some specific information is needed by WebSphere 4.0 in its deployment descriptor. This information is defined using the "*Set specific EJB properties*" command of the EJB menu.

The main dialog box is shown Figure 8-11 below.

Figure 8-11. The window through which information needed by WebSphere 4.0 is entered

This window contains properties which can be set using the assembly tool of IBM's WebSphere 4.0. For further details, please refer to the WebSphere 4.0 documentation.

IONA iPortal Application Server 3.0 and Objecteering/UML

During module configuration, certain fields are used to configure your EARSCO project (as shown in Figure 8-12).

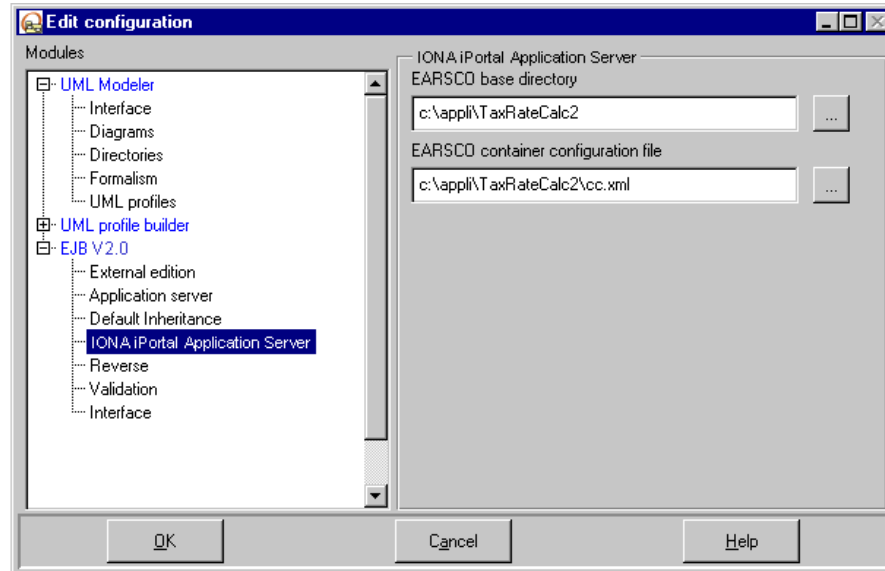


Figure 8-12. The "IONA iPortal Application Server" sub-category of parameters

The "*EARSCO base directory*" field is used to indicate the root directory of your EARSCO project, whilst the "*EARSCO container configuration file*" field specifies the container configuration file you are using to deploy the application.

To insert your EJB into the EARSCO framework and register it in the different XML files, run the "*Run packaging tool*" command.

Sun J2EE Server and Objectteering/UML

After you have created the JAR file with Objectteering/UML, follow the steps below:

- 1 - Select the subsystem of your EJB and run the "*Run packaging tool*" command from the EJB menu.
- 2 - This command will run "*deploy tool*" from Sun J2EE. This tool will help you define options for your EJB's container.
- 3 - Deploy your EJB on the server.

Please refer to Sun's documentation for more details on Sun's deployment tools.

JBoss 3.0 and other application servers

The Objectteering/EJB module can be configured to use other application servers. This section presents the use of JBoss (www.jboss.org) with Objectteering/UML.

When selecting your platform, you can choose “None” as the platform name. This means that you will be using a platform which is not in the default list. You can also select your j2ee.jar file (as shown in Figure 8-13 below).

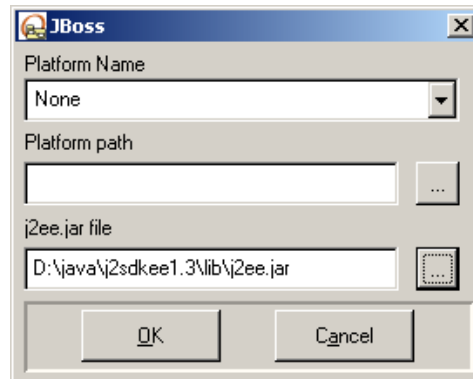


Figure 8-13. Selecting a j2ee.jar file

You must now manually configure the “*Application server*” options in Objectteering/UML (Figure 8-14).

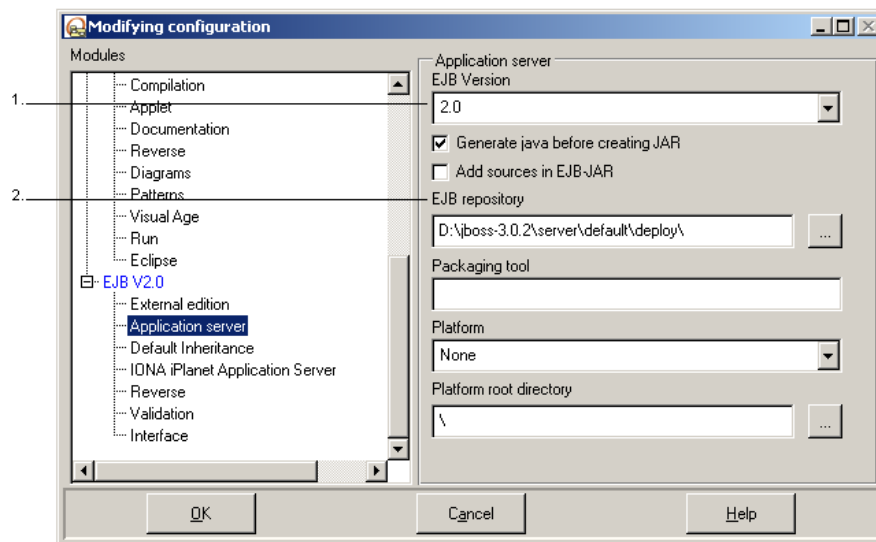


Figure 8-14. Configuring application server options

- 1 - The “*EJB Version*” field is used to define the EJB version supported by the server. JBoss 3.0 supports EJB 2.0.
- 2 - The “*EJB repository*” field is used to indicate the directory where Objectteering/UML will place the generated JAR file. JBoss 3.0 automatically loads EJBs added to its deployment directory, which means you should specify the deployment directory of your server in this field.

Chapter 8: Specific EJB properties

You must now configure your CLASSPATH to add JBoss JAR files. The j2ee.jar file was added through the platform selection box. In the “*Accessible class*” field at Java configuration level, you must add the following (assuming that JBoss is installed in c:\jboss-3.0.2\):

```
C:\jboss-3.0.2\client\jnp-client.jar;C:\jboss-3.0.2\client\jboss-common-client.jar;C:\jboss-3.0.2\client\log4j.jar;C:\jboss-3.0.2\client\jboss-client.jar;C:\jboss-3.0.2\client\jbossx-client.jar
```

When you create a JAR file using the “*Create an EJB JAR*” command on a subsystem, Objectteering/UML creates the corresponding JAR file and copies it into the EJB repository, so in JBoss, it is automatically deployed on the server.

You can also use the “*Packaging tool*” parameter to define a command to be launched with the “*Run packaging tool*” command. For details on this field, please refer to the “*Parameter sets*” section in chapter 7 of this user guide.

If your application server asks for specific Xml files (such as JBoss and its jboss.xml file), you can add them to a note (as shown in Figure 8-15).

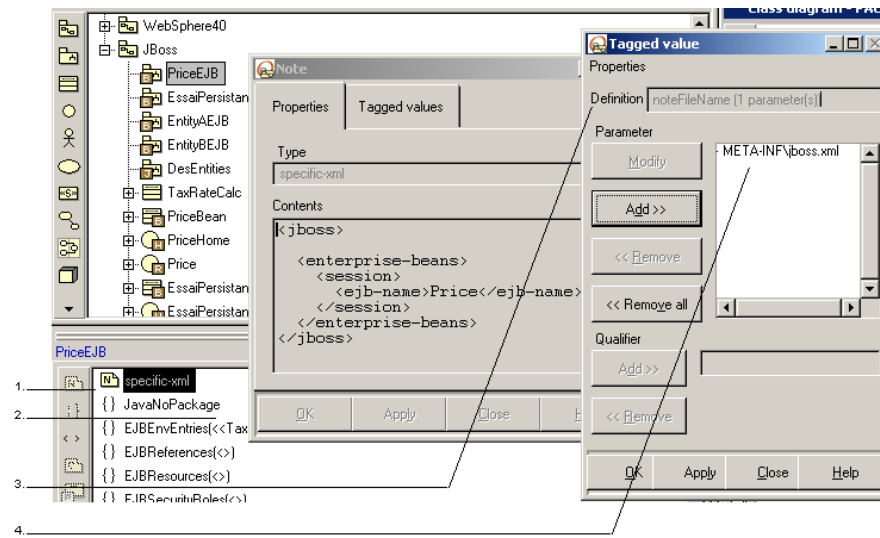


Figure 8-15. Adding specific XML files

Steps:

- 1 - On the subsystem you wish to deploy, create a specificXml type note.
- 2 - Edit this note, and write the contents of your file.
- 3 - To define the filename corresponding to this note, add a {noteFileName} type tagged value.
- 4 - For this tagged value, add as parameter the name of the file, referring to the JAR root. For JBoss, when you create a note for the jboss.xml file, this parameter must have the following value: META-INF\jboss.xml.

Index

"First_Steps" package	3-7	{EJBUncallable} tagged value	6-23
"WebLogic" package	3-7	{JavaFilterAccessor} tagged value	6-19
"WebSphere35" package	3-7	{noteFileName} tagged value	8-29
"WebSphere40" package	3-7	<\$OBJING_PATH>	2-3
{EJBAbstractSchemaName} tagged value	6-25	<< EJBRealizeRemote>>	6-28
{EJBAcknowledgeMode} tagged value	6-25	<<EJBClientJar>>	6-28
{EJBEnvEntries} tagged value	6-10, 6-24	<<EJBCompField>>	6-29
{EJBLocalRef} tagged value	6-10, 6-24	<<EJBCreateMethod>>	6-27
{EJBMessageDrivenDestination} tagged value	6-25	<<EJBDescriptor>>	6-29
{EJBMessageSelector} tagged value	6-25	<<EJBEntityBean>>	6-28
{EJBNameInJar} tagged value	6-24	<<EJBEntityHomeInterface>>	6-27
{EJBPersistenceType} tagged value	6-24	<<EJBFinderMethod>>	6-27
{EJBReentrant} tagged value	6-24	<<EJBImplementation>>	6-27
{EJBReferences} tagged value	6-10, 6-24	<<EJBImplementation>> stereotype	1-6
{EJBResourceRef} tagged value	6-10, 6-24	<<EJB-JAR>>	6-28
{EJBResources} tagged value	6-10, 6-24	<<EJBMessageDrivenBean>>	6-28
{EJBRoleName} tagged value	6-23	<<EJBObjectEntityHomeInterface>>	6-27
{EJBSecurityIdentity} tagged value	6-25	<<EJBObjectInterface>>	6-27
{EJBSecurityRole} tagged value	6-24	<<EJBObjectSessionHomeInterface>>	6-27
{EJBSecurityRoles} tagged value	6-10	<<EJBOnMessage>>	6-27
{EJBSessionType} tagged value	1-6, 6-16, 6-23	<<EJBPrimaryKey>>	6-28
{EJBTransAttribute} tagged value	6-23	<<EJBPrimaryKeyField>>	6-29
{EJBTransType} tagged value	6-24	<<EJBRealizeHome>>	6-28
		<<EJBRelationship>>	6-29
		<<EJBRemoteInterface>>	6-27
		<<EJBRemoteMethod>>	6-27
		<<EJBSessionBean>>	6-28
		<<EJBSessionHomeInterface>>	6-27
		<<JavaArchiveFile>>	6-28
		<<JavaClassFile>>	6-29
		Abstract schema name	6-10

- Add sources in EJB JAR 7-7
- Application server parameters 7-3, 7-6
- Application servers 8-26
- AssociationEnd 6-19
- Attribute map 8-8
- Attribute name 8-8
- Bean class 1-7, 6-6
- Bean Managed Persistence 6-16
- Cache strategy 8-12
- Cache type 8-12
- Cardinality 6-18
- Check class 6-17
- Check class for EJB conversion 6-17
- Check EJB 6-13
- Check operation 6-22
- Checking, generation and validation rules 4-3
- Choosing a bean 3-17, 6-6
- Choosing a platform 2-6, 3-8
- Class functions 6-15
- Class to create 6-5
- Clusters properties dialog box 8-15
- Code/model consistency 1-4
- Column name 8-8
- Command for invoking external editor 7-5
- Commands
 - The "Set specific EJB properties" command 8-23
- Commands available on a class 6-15
 - The "Check class for EJB conversion" command 6-17
 - The "Check class" command 6-17
 - The "Convert to an EJB" command 6-16
- The "Create a container managed relationship" command 6-18
- The "Set EJB properties" command 6-16
- The "Update the EJB with this class" command 6-16
- Commands available on a package 6-3
 - The "Check EJB" command 6-13
 - The "Create an EJB subsystem" command 6-6
 - The "Create EJB classes" command 6-4, 6-16
 - The "Reference EJB in EJB-JAR" command 6-7
- Commands available on a subsystem 6-8
 - Run packaging tool 3-9
 - The "Create an EJB JAR" command 6-11, 8-28
 - The "Delete an EJB subsystem" command 6-12
 - The "Run packaging tool" command 6-11, 8-3, 8-22, 8-24, 8-25, 8-28
 - The "Set EJB properties" command 6-9
- Commands available on a subsystem and a package
 - The "Reverse an EJB JAR" command 6-13
 - The "Select J2EE platform" command 6-14
- Commands available on an operation 6-20
 - The "Check operation" command 6-22
 - The "Delete an operation from the EJB" command 6-22
 - The "Set EJB properties" command 6-21

The "Set operation create" command	6-22	Creating an EJB relationship	6-18
The "Set operation finder" command	6-22	Creating an EJB subsystem	3-16
The "Set operation remote" command	6-22	Creating the EJB JAR file	3-23
Configuring application server options	8-27	Creating the EJB structure	3-13
Configuring the Objecteering/EJB module	7-4	Customizing the EJB generator	4-3
Application server parameters	7-6	Data source name	8-8
Default inheritance parameters	7-8	Database properties dialog box	8-7
External edition parameters	7-5	DB is shared	8-14
Interface parameters	7-12	Default inheritance parameters	7-3, 7-8
IONA iPortal Application Server parameters	7-9	Defining module parameters	
Reverse parameters	7-10	Application Server options	7-3
Validation parameters	7-11	Default inheritance	7-3
Consistency	4-4	External edition options	7-3
Consistency checks	4-6	Interface options	7-3
Container	1-7	IONA iPortal Application Server options	7-3
Container Managed Persistence	6-16	Reverse options	7-3
Converting a class into an EJB	5-3	Validation options	7-3
Converting a package to an EJB-JAR	6-7	Delay updates until end of tx	8-14
Converting to an EJB	3-13	Delete an operation from the EJB	6-22
Creating a container managed relationship	6-18	Deleting an EJB subsystem	6-12
Creating a JAR from an EJB	6-11	Deployer tool	1-7, 3-9, 6-11
Creating a Java generation work product	3-19	Deploying the EJB on your application server	3-25
Creating a new environment entry	3-18, 6-10	Deployment descriptor	8-4, 8-23
Creating a UML modeling project	2-4	Description	8-19
Creating a UML modeling project for developing EJB applications	2-4	Destination	6-18
Creating an EJB	3-15, 6-5	Destination entity	6-18
		Developing an EJB component	
		Creating a JAR file	1-6
		Creating a UML model	1-6
		Deploying a JAR file on your application server	1-6
		Documents	4-3
		EARSCO base directory	7-9, 8-24

- EARSCO container configuration file 7-9, 8-24
- EARSCO project 8-24
- Editing the configuration of the application server 3-9
- EJB 1-7
- EJB component generation 1-3
- EJB demonstration project 3-3
- EJB first steps 1-3
- EJB module parameters 4-3
- EJB name 6-4, 6-6, 8-6, 8-19
- EJB name in JAR 6-9
- EJB properties
 - EJBEnvEntries 3-16
- EJB repository 7-7, 8-27
- EJB repository path 3-9
- EJB resources 6-10
- EJB skeleton 3-3
- EJB type 3-14, 6-4, 6-16
 - Entity 6-4
 - Message driven 6-4
 - Session stateful 6-4
 - Session stateless 3-14, 6-4
- EJB version 7-7, 8-27
- EJBFinderMethods 6-16
- EJBQL 6-26
- EJBQL query 6-21
- EJB's reference name 8-17
- Enable call by reference 8-12
- Entity 1-7
- Environment entries 6-10
- Expression list 8-10
- External edition parameters 7-3, 7-5
- Field name 6-18
- Find for update 8-10
- Finder call ejbLoad 8-14
- Finder operation 6-21
- Finder properties dialog box 8-9
- Finders properties 8-21
- First steps
 - Configuring the Objectteering/EJB module 3-8
 - Converting an existing class into a session stateless EJB 3-3
 - Creating a Java generation work product 3-19
 - Creating an EJB subsystem 3-16
 - Creating the EJB JAR file 3-23
 - Creating the EJB structure 3-13
 - Deploying the EJB on your application server 3-25
 - Generating the Jar file containing the EJB 3-3
 - Importing a model into the first steps project 3-5
 - Initializing the first steps project 3-3
 - Modifying an operation 3-11
- First steps preparation
 - Creating a UML modeling project 3-4
 - Launching Objectteering/UML Modeler 3-4
 - Selecting the Objectteering/EJB module 3-4
 - Selecting the Objectteering/Java module 3-4
- General first steps 1-3
- General Objectteering/UML first steps 3-3
- Generate identifiers 7-5
- Generate java before creating JAR 7-7
- Generation work product 4-4
- Home call router class name 8-16

- Home interface 1-7
- Home is clusterable 8-16
- Home load algorithm 8-16
- IBM
 - jetace 8-22
- Idle timeout in seconds 8-12
- Importing a model into the first steps project 3-5
- Importing the EJB first steps project 3-5
- Inherit from 6-5
- Initial beans in free pool 8-12
- Inserting your EJB into the EARSCO framework 8-24
- Installation directories 2-3
- Installing the Objectteering/EJB module 2-3
- Interface parameters 7-3, 7-12
- IONA iPortal application server parameters 7-3, 7-9
- Is-modified method name 8-14
- Isolation level 8-8, 8-19
- Items 4-3
- J methods for producing EJB code zones 4-3
- j2ee.jar file 6-14, 8-26
- Jar file 3-3
- JAR file to reverse 6-13
- JAR root 8-29
- Java generation work product 3-19
- java package 3-4
- javax package 3-4
- JBoss 8-26
- JBoss JAR files 8-28
- jboss.xml file 8-29
- JCP 6-23
- jetace 8-22
- JNDI name 8-6
- License 2-3
- Local object 6-4
- Local references 6-10
- Log file 7-11
- Max beans in cache 8-12
- Max beans in free pool 8-12
- Method interface 8-19
- Method name 8-10, 8-19
- Method parameters 8-10
- Model consistency 4-6
- Model-driven generation 1-4
- Modifying an operation 3-11
- Modifying module parameter configuration 3-9
- Module parameters 1-4
 - Application server path 1-4
 - Default inheritance 1-4
 - External edition options 1-4
 - IONA iPortal application server options 1-4
 - Packaging tool command line 1-4
 - Reverse options 1-4
 - Validation log file path 1-4
- Naming your EJB 3-17, 6-6
- Notes 4-3, 4-5
 - EJBQL 6-26
- Objectteering/Administrating
 - Objectteering/UML Sites 1-3
- Objectteering/EJB functions 1-3
 - Checking the validity of your EJB model 1-3
 - Converting a UML class to an EJB 1-3
 - Creating a new EJB 1-3
 - Creating and deploying a platform independent JAR file 1-3

Creating standard modeling of your EJBs	1-3	References	6-10
Reversing an existing EJB	1-3	References list	8-17
Objectteering/Introduction	1-3, 2-3, 3-3, 4-3	References properties dialog box	8-17
Objectteering/Java	2-3, 3-19	Relation name	6-18
Objectteering/UML consistency checks	4-6	Remote interface	1-7
Objectteering/UML installation	2-3	Removable consistency checks	4-6
Objectteering/UML Modeler	1-3, 2-4, 4-6	Resource env ref	6-10
Operation functions	6-20	Resource's JNDI name	8-17
Optimization dialog box	8-11	Resources list	8-17
Package functions	6-3	Resource's reference name	8-17
Packaging tool	7-7	Resources	6-10
Parameter type	8-19	Result type mapping	6-21
Parameterization	1-4	Reverse feature	1-3
Parameterizing the Objectteering/EJB module	4-3	Reverse only XML	7-10
EJB module parameters	4-3	Reverse parameters	7-3, 7-10
Tagged values, notes and stereotypes	4-3	Reversing an EJB JAR	6-13
UML Profile Builder tool	4-3	Role name list	8-21
Parameter's list	8-19	Run packaging tool	6-11
Passivation strategy	8-12	Schema name	8-8
Persistence properties dialog box	8-13	SDK Enterprise Edition	2-3
Persistence type	8-14	SDK Standard Edition	2-3
Platform name	6-14	Security identity	6-9
Platform path	6-14	Security role	1-7
Platform root directory	7-7	Security roles	6-10, 6-21
Prerequisites to working with the Objectteering/EJB module	2-3	Security roles dialog box	8-20
Primary key class	1-7, 5-3, 6-5	Selecting a platform	6-14, 8-26
Principal name list	8-21	Selecting a subsystem to reference	6-7
Properties editor	4-4	Selecting the Objectteering/EJB module	2-4, 4-3
Read timeout in seconds	8-12	Selecting the Objectteering/EJB module for the new UML modeling project	2-5
Reference's JNDI name	8-17	Session EJB session type property	6-16
		Session Stateful	1-7

Session Stateless	1-7	<<EJB-JAR>> stereotype	6-28
Session stateless EJB	3-3	<<EJBMessageDrivenBean>>	
Set create	5-3	stereotype	6-28
Set finder	5-3	<<EJBObjectEntityHomeInterface>>	
Set for persistence	5-3	stereotype	6-27
Set for primary key	5-3	<<EJBObjectInterface>> stereotype	6-27
Set operation create	6-22	<<EJBObjectSessionHomeInterface>>	
Set operation finder	6-22	>> stereotype	6-27
Set operation remote	6-22	<<EJBOnMessage>> stereotype	6-27
Set remote	5-3	<<EJBPrimaryKey>> stereotype	6-28
Setting EJB properties	6-16, 6-21	<<EJBPrimaryKeyField>>	
Source	6-18	stereotype	6-29
specificXml notes	8-3	<<EJBRealizeHome>> stereotype	6-28
Stateful session persistent store		<<EJBRealizeRemote>> stereotype	6-28
directory	8-14	<<EJBRelationship>> stereotype	6-29
Stateless bean call router class name		<<EJBRemoteInterface>>	
	8-16	stereotype	6-27
Stateless bean is clusterable	8-16	<<EJBRemoteMethod>> stereotype	6-27
Stateless bean load algorithm	8-16	<<EJBSessionBean>> stereotype	6-28
Stateless bean methods are		<<EJBSessionHomeInterface>>	
idempotent	8-16	stereotype	6-27
Stereotypes	1-6, 4-3, 4-5	<<JavaArchiveFile>> stereotype	6-28
<<EJBClientJar>> stereotype	6-28	<<JavaClassFile>> stereotype	6-29
<<EJBCompField>> stereotype	6-29	Stereotypes on a class	6-27
<<EJBCreateMethod>> stereotype	6-27	Stereotypes on a component	6-29
<<EJBDescriptor>> stereotype	6-29	Stereotypes on a link	6-28
<<EJBEntityBean>> stereotype	6-28	Stereotypes on a package	6-28
<<EJBEntityHomeInterface>>		Stereotypes on a subsystem	6-28
stereotype	6-27	Stereotypes on an association	6-29
<<EJBFinderMethod>> stereotype	6-27		
<<EJBImplementation>>	1-6		
<<EJBImplementation>> stereotype	6-27		

- Stereotypes on an attribute 6-29
- Stereotypes on an operation 6-27
- Subsystem functions 6-8
- Sun deployment tools 8-25
- Sun J2EE 8-25
- Table name 8-8
- Tagged value parameters
 - Stateful 1-6
 - Stateless 1-6
- Tagged values 1-4, 1-6, 4-3, 4-5
 - {EJBAbstractSchemaName} tagged value 6-25
 - {EJBAcknowledgeMode} tagged value 6-25
 - {EJBEnvEntries} tagged value 6-10, 6-24
 - {EJBLocalRef} tagged value 6-10, 6-24
 - {EJBMessageDrivenDestination} tagged value 6-25
 - {EJBMessageSelector} tagged value 6-25
 - {EJBNameInJar} tagged value 6-24
 - {EJBPersistenceType} tagged value 6-24
 - {EJBReentrant} tagged value 6-24
 - {EJBReferences} tagged value 6-10, 6-24
 - {EJBResourceRef} tagged value 6-10, 6-24
 - {EJBResources} tagged value 6-10, 6-24
 - {EJBResultTypeMapping} tagged value 6-23
 - {EJBRoleName} tagged value 6-23
 - {EJBSecurityIdentity} tagged value 6-25
 - {EJBSecurityRole} tagged value 6-24
 - {EJBSecurityRoles} tagged value 6-10
 - {EJBSessionType} tagged value 1-6, 6-16, 6-23
 - {EJBTransAttribute} tagged value 6-23
 - {EJBTransType} tagged value 6-24
 - {EJBUncallable} tagged value 6-23
 - {JavaFilterAccessor} tagged value 6-19
 - {noteFileName} tagged value 8-29
- Tagged values on a home interface class 6-23
- Tagged values on a subsystem stereotyped EJBEnterpriseBean 6-24
- Tagged values on an operation 6-23
- Testing your EJB 3-25
- The "Convert to an EJB" command 3-13
- The "Create an EJB JAR" command 3-24
- The "Create an EJB" window 3-15
- The "Generate and compile" command 3-22
- The "Set EJB properties" window 3-18
- Transaction isolation 8-21
- Transaction isolation dialog box 8-18
- Transaction timeout in seconds 8-12
- Transaction type 6-9
- Type identifier in Persistence type 8-14
- Type identifier in Persistence use 8-14

Type storage in Persistence type	8-14	Validate EJB before packaging	7-11
Type version in Persistence type	8-14	Validation log file	3-10
Type version in Persistence use	8-14	Validation parameters	7-3, 7-11
UML model root	3-5	WebLogic 6.0	8-4
UML Profile for EJB	6-23	Weblogic 7.0	8-3
Update the EJB with this class	5-4, 5-5	WebLogic builder	8-3
Using existing EJB components	1-3	WebLogic version	8-6
Using the module	2-4	WebSphere 4.0	8-23
		Work products	
		Description	4-4
		XML files	8-4, 8-24