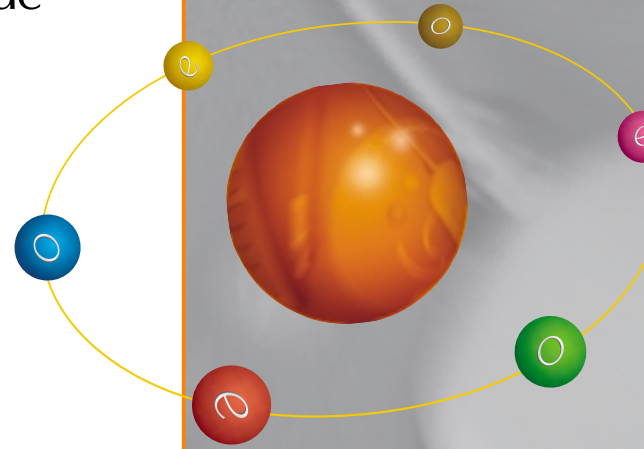


Objecteering/UML

Objecteering/UML Teamwork User Guide

Version 5.2.2



Objecteering

Software

www.objecteering.com

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 Objecteering 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 Objecteering Software.

© 2002 Objecteering Software

Objecteering/UML version 5.2.2 - CODOBJ 001/001

Objecteering/UML is a registered trademark of Objecteering 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 to teamwork with Objectteering/UML | |
| Teamwork in Objectteering/UML | 1-3 |
| Structure of the Objectteering/UML teamwork user guide | 1-7 |
| Physical organization | 1-8 |
| Migrating the repository | 1-12 |
| Warnings and restrictions | 1-15 |
| Glossary | 1-18 |
| Chapter 2: Functions of the Objectteering/UML teamwork modules | |
| Overview of teamwork module commands | 2-3 |
| Commands available on elements in read-only mode | 2-7 |
| Commands available on elements in read-write mode | 2-9 |
| Commands available on all elements (read-only and read-write)..... | 2-13 |
| Chapter 3: Typical usage | |
| Overview of typical teamwork module usage | 3-3 |
| Defining the initial model..... | 3-4 |
| Multi-user atomic units | 3-6 |
| Links | 3-8 |
| How should I proceed after a problem? | 3-11 |
| Managing work products and generated files | 3-12 |
| Chapter 4: Transferring elements between UML modeling projects | |
| Running principle | 4-3 |
| Importing elements between UML modeling projects | 4-5 |
| Chapter 5: Parameterizing the Objectteering/UML teamwork modules | |
| Defining module parameters | 5-3 |
| Graphic representation of the read-only and read-write modes | 5-7 |
| The administration module | 5-9 |
| Chapter 6: The Objectteering/Multi-user module | |
| Introduction to the Objectteering/Multi-user module | 6-3 |
| Using the Objectteering/Multi-user module | 6-4 |
| Objectteering/Multi-user commands..... | 6-10 |

| | |
|---|------|
| Chapter 7: The Objecteering/ClearCase module | |
| Introduction to the Objecteering/ClearCase module | 7-3 |
| Using the Objecteering/ClearCase module | 7-5 |
| Objecteering/ClearCase commands | 7-11 |
| Effects of commands in ClearCase | 7-13 |
| Chapter 8: The Objecteering/SCC module | |
| Introduction to the Objecteering/SCC module | 8-3 |
| Using the Objecteering/SCC module | 8-5 |
| Objecteering/SCC commands | 8-9 |
| Objecteering/SCC module parameters | 8-16 |
| Index | |

Chapter 1: Introduction to teamwork with Objecteering/UML

Teamwork in Objectteering/UML

Overview of teamwork in Objectteering/UML

Welcome to the *Objectteering/UML teamwork user guide!*

The *Objectteering/UML Modeler* tool works in the context of an Objectteering UML modeling project. All the data from this UML modeling project is located in the same physical location (an .ofp file) and only one user can work on it at a time.

However, when it comes to teamwork, it is necessary:

- ◆ for several people to work together at the same time
- ◆ to manage work spaces, such as development or integration spaces
- ◆ to exchange model information between different UML modeling projects in a coordinated manner

A certain level of knowledge in group work is necessary, and you must be familiar with the Objectteering/UML environment, detailed in the following user guides:

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

We recommend that all users carry out the Objectteering/UML first steps project in the *Objectteering/Introduction* user guide, before starting teamwork activities, in order to become sufficiently at ease with the various general functions provided by Objectteering/UML.

Teamwork modules

To meet all your teamwork needs, Objectteering/UML provides the following teamwork modules:

- ◆ The *Objectteering/Multi-user* module, which is only available in the Enterprise Edition of Objectteering/UML, is used to simply support teamwork operations
- ◆ The *Objectteering/ClearCase* module, which is only available in the Enterprise Edition of Objectteering/UML, is used to couple Objectteering/UML with Rational ClearCase, and to carry out the most common ClearCase operations on the Objectteering/UML model
- ◆ The *Objectteering/SCC* module, which is only available in the Enterprise Edition of Objectteering/UML and exclusively for the Microsoft Windows environment, is used to couple Objectteering/UML to different version control systems which use the Microsoft SCC API.

Please note that it is not possible (and would be pointless anyway) to use several of these modules at once, as each module provides the same commands. They are not complementary, but are rather functionally independent.

Work spaces in Objecteering/UML

A site is where the work carried out in a company or in one of its departments is concentrated. A site can contain several UML modeling projects.

Several "multi-user projects" can exist on a site. These are defined and managed by the *Objecteering/Multi-user*, *Objecteering/ClearCase* or *Objecteering/SCC* modules. These multi-user projects are themselves broken down into UML modeling projects, which make up Objecteering/UML Modeler development and user work spaces.

Within a multi-user project, communication between development spaces is managed by the *Objecteering/Multi-user* module.

On a site, the exchange of information (see Figure 1-1) between multi-user projects is carried out in two different ways:

- ◆ The import of elements between UML modeling projects
- ◆ The exchange of externalized data through *Objecteering/Multi-user*, *Objecteering/ClearCase* or *Objecteering/SCC*.

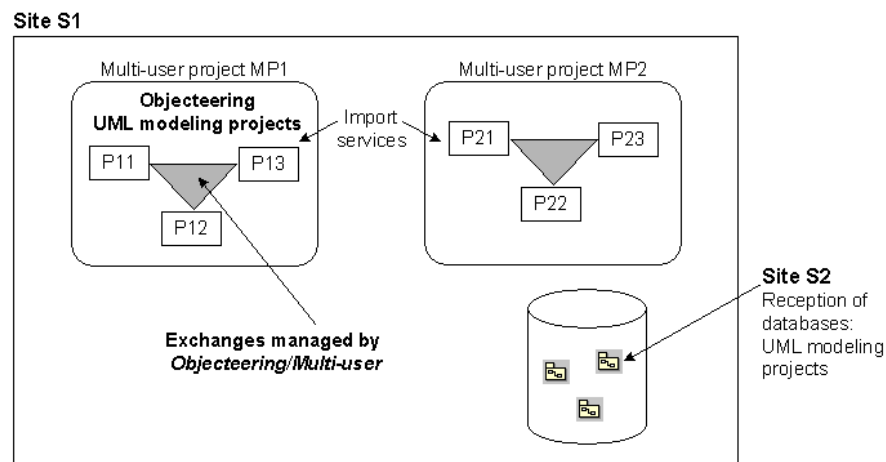


Figure 1-1. Exchanges between work spaces in Objecteering/UML

Model exchange facility

In addition to the standard teamwork services provided, there also exist model exchange functions:

- ◆ the import service between databases. This service, described in chapter 4 of this user guide, allows you to use the Objectteering/UML universal model identification mechanism to carry out exchanges between UML modeling projects and databases. This service is only available in the Enterprise edition of Objectteering/UML.
- ◆ the XMI exchange service. The standard XMI format is used to exchange models between different UML modeling tools. Whilst not as powerful as the internal Objectteering/UML formats (no identification mechanism, no exchange of diagrams, etc.), this service is available in the Personal Edition and the Professional Edition of Objectteering/UML, subject to certain restrictions ("complete" UML modeling project exchange only). For further information on XMI, please refer to the *Objectteering/XMI* user guide.

Creating "*multi-user projects*" using an Objectteering/UML teamwork module necessitates a certain level of discipline regarding the use of the module in question. At the start of development, the user can easily content himself with exchange services.

Structure of the Objecteering/UML teamwork user guide

The *Objecteering/UML teamwork user guide* is structured as follows:

- ◆ Chapter 1 - "*Introduction to teamwork with Objecteering/UML*": This chapter presents the different Objecteering/UML teamwork modules.
- ◆ Chapter 2 - "*Functions of the Objecteering/UML teamwork modules*": This chapter describes the commands provided by the Objecteering/UML teamwork modules.
- ◆ Chapter 3 - "*Typical usage*": This chapter presents typical use of the Objecteering/UML teamwork modules.
- ◆ Chapter 4 - "*Transferring elements between UML modeling projects*": This chapter describes the transfer of different models elements between UML modeling projects.
- ◆ Chapter 5 - "*Parameterizing the Objecteering/UML teamwork modules*": This chapter presents the different module parameters which manage the behavior of the services provided.
- ◆ Chapter 6 - "*The Objecteering/Multi-user module*": This chapter describes the specific behavior of the Objecteering/Multi-user module.
- ◆ Chapter 7 - "*The Objecteering/ClearCase module*": This chapter describes the specific behavior of the Objecteering/ClearCase module.
- ◆ Chapter 8 - "*The Objecteering/SCC module*": This chapter describes the specific behavior of the Objecteering/SCC module.

Chapters 1-5 concern all Objecteering/UML teamwork modules and provide information which will be of use to all users.

Chapters 6, 7 and 8 discuss the specificities of each Objecteering/UML teamwork module. Users need only refer to the chapter describing the teamwork module they are using.

Physical organization

Introduction

This section presents the organization of the directories and files placed in the multi-user work space dedicated to teamwork.

Principle

The *Objecteering/UML* teamwork modules are based on the internalization/externalization mechanism managed by Objecteering/UML. The externalization function is used to generate ASCII files which are representative of modeling elements (packages, classes, actors, etc) from the contents of an Objecteering/UML modeling project. Conversely, the internalization function is used to reread ASCII files, in order to update the information present in an Objecteering/UML modeling project.

In the suggested organization, each user has an Objecteering/UML database in which a personal work project is defined. In this way, the user models in a private work space, whilst being able to reserve elements (check-out), deliver them (check-in) or quite simply update his personal model with regard to the multi-user model (Import, ...).

The multi-user model, which is made up of ASCII files, also allows the version and configuration management of modeling elements to a high level of precision (packages, classes, actors, use cases, ...).

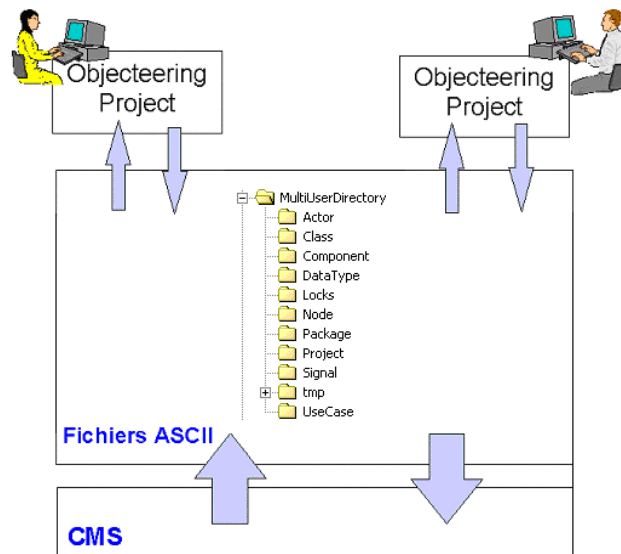


Figure 1-2. The multi-user work space

Directories

The directory which represents the repository is made up of several levels of information directly linked to module possibilities:

- ◆ A directory for each type of multi-user atomic unit
- ◆ A "*tmp*" directory used to store temporary files used during optimization phases, present only with the *Objectteering/Multi-user* teamwork module
- ◆ A "*Locks*" directory containing locking files, used in the locking of elements or of the repository itself
- ◆ A "*Deleted*" directory, present only with the *Objectteering/Multi-user* module and containing elements explicitly destroyed in Objectteering/UML Modeler. The files corresponding to destroyed elements are not deleted from the repository, but are only moved.
- ◆ A "*root*" directory containing each user's log file (*username.log*). The location and name of the log can be modified during module configuration. For further information on module parameterization, please refer to chapter 5, "*Parameterizing the Objectteering/UML teamwork modules*", of this user guide.

Note: It should be noted that the file which manages information relative to model elements has a name which is internal to Objectteering/UML (principally constructed from element identifiers).

The location of Objectteering/UML modeling projects

We recommend that users create their databases on their machine's local disk, rather than a remote disk, as teamwork operations, as well as the use of Objectteering/UML itself, are much faster in this case. The Objectteering/UML modeling project is simply a view of the repository, which must be shared and regularly archived.

Using the externalization binary

The *obj_extreport* binary is used to find the list of the model's objects through an externalization hierarchy. To launch this binary, use the following syntax:

```
obj_extreport<CompleteExternalizationPath>  
[<TargetReportFileName>]
```

Example

Figure 1-3 below presents an example of objects listed in an externalization hierarchy.

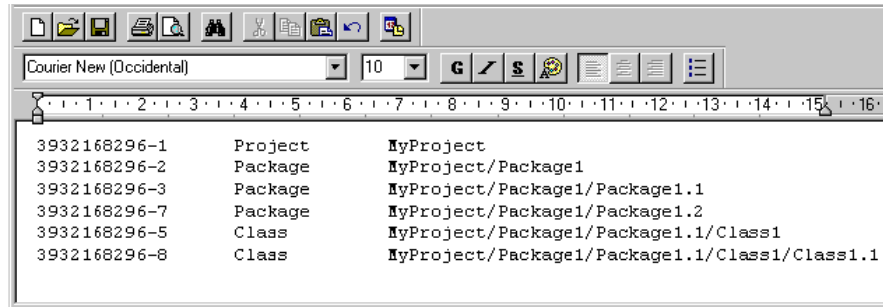


Figure 1-3. Example of externalized objects in an externalization hierarchy

Migrating the repository

Introduction

When migrating Objecteering/UML from an earlier version, it is important to correctly carry out the migration of Objecteering/UML models which use one of the Objecteering/UML teamwork modules, so as to guarantee that the teamwork repository will be compatible with the version of the module delivered with Objecteering/UML 5.2.2.

Two case scenarios are possible:

- ◆ migration from version 5.2.1 of Objecteering/UML
- ◆ migration from an earlier version of Objecteering/UML

Before carrying out any migration operations, we recommend that you save your UML models (.ofp files).

If you are using the *Objecteering/Multi-user* module, we recommend that you save the repository.

If you are using the *Objecteering/ClearCase* or *Objecteering/SCC* modules, we recommend that you apply a label to the teamwork repository before carrying out migration operations. This application of a label must only be carried out if all files have been checked-in. In this case, all users should run a "*Project check-in*" before applying the label. After this operation, an "*Import from repository*" must be run on an Objecteering/UML model, so as to transfer the latest corrections made by other users. It is then possible to run the "*Apply label*" command on the root of the Objecteering/UML model.

Migration from version 5.2.1 of Objecteering/UML

In the case of migration from version 5.2.1 of Objecteering/UML, no particular migration operations are necessary.

After updating your installation to version 5.2.2, you should simply select, for each database, the new version of the Objecteering/UML teamwork module you are using. For details on migrating databases, please refer to the "*Stand alone installation in Windows*" section in chapter 2 of the *Objecteering/Introduction* user guide.

The existing teamwork repository is managed by the new version of the Objecteering/UML teamwork module. User names and directories defining the repository remain valid.

Note: Module parameter settings are retained when the latest version of an Objecteering/UML teamwork module is selected.

Migration from an earlier version of Objectteering/UML

In the case of migration from a version of Objectteering/UML earlier than the 5.2.1 version, the following steps should be carried out:

- 1 - For each Objectteering/UML model, run a check-in on all elements which have been checked-out , so as to guarantee that the repository contains the latest modifications. This operation can be launched by running the "*Project check-in*" command for each Objectteering/UML model.
 - 2 - From an Objectteering/UML model, run the "*Import from repository*" command, in order to have a complete Objectteering/UML model, and then unselect the teamwork module. Only this Objectteering/UML model will exist after site migration. All the other Objectteering/UML models will no longer be used after installation of the Objectteering/UML 5.2.2.
 - 3 - After ensuring that you have the correct license file for Objectteering/UML 5.2.2 and stopping the "*objingsrv*" and "*lmgrd*" services, launch the update of your site.
 - 4 - After completing the update of your Objectteering/UML site and restarting the "*objingsrv*" and "*lmgrd*" services, the Objectteering/UML 5.2.2 server is operational.
 - 5 - All client workstations should be updated, by selecting the new modules delivered with version 5.2.2, including the teamwork module.
 - 6 - On the Objectteering/UML model used during phase 2, select the teamwork module, and define the name and directory in which the new teamwork repository will be created.
 - 7 - A new database should then be created for each user from the database used to create the new teamwork repository. It is essential that the "*Copy a database*" command in the Objectteering/UML database administration tool be used.
 - 8 - For each database, select the administration module, to define the user fields with the correct person and then select this module, before making the model available.
-

Warnings and restrictions

Introduction

All the Objecteering/UML teamwork modules are subject to certain restrictions, which are presented in this chapter. The restrictions described are common to all the teamwork modules. For restrictions specific to each module, please refer to the related chapter in this user guide.

Objecteering/UML versions

All users connected to a repository must be using the same version of Objecteering/UML and the same version of the teamwork module.

Undo/Redo



During *check-in*, *check-out* and *undo check-out* operations, a backup of the UML modeling project is made. The "*Undo/Redo*" commands are not, therefore, available.

Consistency checks

A consistency check is run within the Objectteering/UML modeling project, but not within the repository. It is, therefore, possible to inadvertently create inconsistencies within the model you wish to import. In order to detect possible consistency problems within the repository as soon as possible, we recommend that you regularly execute the "*Import from repository*" command on the UML modeling project, in order to import the entire contents of the repository, thus triggering a consistency checking operation on the UML modeling project itself.

To guarantee the consistency of models built using Objectteering/UML, checks are continually run by the tool during modeling and internalization phases. We strongly recommend against deactivating consistency checks, and it is imperative that you NEVER publish ("Check-in") model units in the repository without them being active.

As a reminder:

- ◆ The  icon indicates that consistency checks are active.
- ◆ The  icon indicates that consistency checks are deactivated.

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.

User

For the *Objectteering/Multi-user* module, it is important not to configure two Objectteering/UML modeling projects with the same user name. Locking checks ("*Check-out*") are carried out from this name, and if several simultaneously active Objectteering/UML modeling projects have been configured with the same user name, then incorrect processing will be carried out by Objectteering/UML.

The current version of the module carries out no checks on the occurrence of the repetition of user names.

For the *Objectteering/ClearCase* and *Objectteering/SCC* modules, it is imperative that each user have a different login.

Modules used

Each private Objectteering/UML modeling project must have available the same modules (for example, the Objectteering/UML Java or C++ code generation modules). These modules must be selected in each user's UML modeling project during this initialization phase, through the "*Modules*" command of the "*Tools*" menu.

For performance-related reasons, we recommend that a template database be created. This template database should contain all pre-installed and pre-selected modules. If this is done, connection to a repository when creating a new database is more efficient. The teamwork module can be pre-installed, but must not be selected. For further information on this point, please refer to the "*Create a UML model type*" command described in the "*Detailed view of the Administration menu*" section in chapter 3 of the *Objectteering/Administrating Objectteering Sites* user guide.

All users connected to the same repository must use the same teamwork module. For example, it is not possible to have some users using the *Objectteering/Multi-user* module, whilst other users are using the *Objectteering/ClearCase* module.

Stopping commands

As soon as a command has been launched, it is not possible to stop it.

A command run on a large number of elements can take several minutes to complete. In this case, we strongly recommend against stopping Objectteering/UML abruptly, as certain data could be lost if it has not been saved either in the repository or in the Objectteering/UML modeling project.

If Objectteering/UML is stopped abruptly, the procedures described in the "*How should I proceed after a problem?*" section in chapter 3 of this user guide should be followed.

Launching Objectteering/UML directly on a UML modeling project

When launching Objectteering/UML, it is possible to specify the name of the UML modeling project in the command line:

```
objing [<database name>] [<UML modeling project name>]
```

In order not to have to make selections, and to avoid all errors, we recommend that you use this possibility via a UNIX script or a Windows shortcut.

Glossary

- ◆ *API*: Application Protocol Interface. Interfaces between application layers and communicating layers defined in the ISO model.
- ◆ *CMS*: Configuration management system. Application used to store different versions of files, to keep a file history and to retrieve the state of these files as they were on a given date. This tool is used in group work.
- ◆ *Check-in*: The restitution of reserved elements.
- ◆ *Check-out*: The reservation of shared elements in the repository. Operation used to request the modification of model elements. An element cannot be modified by several users at one time, and to this end a locking mechanism managed by the CMS (configuration management system) checks requests for modification.
- ◆ *Inter-project import*: Transfer of data between Objectteering/UML modeling projects, without going through a repository.
- ◆ *Label*: A version label which allows you to associate an identifier to an important version of an element.
- ◆ *Teamwork module*: Module allowing a model to be shared, so that teamwork can be carried out on a UML modeling project. *Objectteering/Multi-user*, *Objectteering/ClearCase* and *Objectteering/SCC* are teamwork modules.
- ◆ *Multi-user atomic unit*: Modeling element which is stored in a file and on which module commands are available. Multi-user atomic units are packages, classes, actors, use cases, signals, components, nodes and data types. Other types of element are not multi-user atomic units and are externalized in the component multi-user atomic unit file.
- ◆ *Objectteering/UML modeling project*: Often called "*project*", this corresponds to the *Objectteering/UML Modeler* area of functioning, and to a user's work area.
- ◆ *Read-only mode*: Mode in which an element may not be modified in any way. An element is put in read-only mode after a check-in operation.

- ◆ *Read-write mode*: Mode in which an element may be modified. An element is put in read-write mode after a check-out operation.
 - ◆ *Repository*: Area containing data shared between several developers.
 - ◆ *SCC*: Source Code Control. Unified interface common to several CMS (configuration management systems) (*VSS*, *PVCS*, etc). This interface only exists in the Windows environment.
 - ◆ *SCC provider repository*: Database internal to a CMS (configuration management system), where information is stored.
 - ◆ *User*: A person identified by a name, and whose Objecteering/UML modeling project constitutes a work area.
-

Chapter 2: Functions of the
Objecteering/UML
teamwork modules

Overview of teamwork module commands

General description

The *Objecteering/Multi-user*, *Objecteering/ClearCase* and *Objecteering/SCC* teamwork modules contain certain commands which can be run from an element's context menu. This context menu is accessed by right-clicking over the element in question.

Three types of command exist:

- ◆ those commands available on elements in read-only mode ("*Check-out*", "*Import from repository*")
- ◆ those commands available on elements in read-write mode ("*Check-in*", "*Check-in and check-out*", "*Undo check-out*")
- ◆ those commands available on all elements, in both read-only and read-write modes ("*Project check-in*")

The *Objecteering/Multi-user*, *Objecteering/ClearCase* and *Objecteering/SCC* modules also provide additional module-specific commands. For details on these commands, please refer to the corresponding chapter in this user guide.

Multi-user atomic units

A multi-user atomic unit is the minimum amount of information which can be reserved. Teamwork commands can only be run on multi-user atomic units.

Multi-user atomic units are instances of the following metaclasses: Project, Package, Class, Actor, Component, Node, UseCase, DataType, Signal.

Other types of model element (attributes, operations, instances, etc.) attached to the multi-user atomic unit are managed along with the multi-user atomic unit itself. Thus, class operations or attributes are information attached to the class. To modify or add an operation to a class, the class in question must be reserved. When the class itself is archived, attributes and operations which may have been modified are also automatically archived.

To delete a multi-user atomic unit, the unit itself, as well as its embedding unit, must be reserved. The embedding unit is modified, since its composition link is altered.

Hierarchical mode

The hierarchical mode allows you to carry out reservations and imports, whilst taking into account model composition hierarchy. This mode is selected by default, in order to facilitate the consistent update of models. However, users must be careful when reserving units, as this reservation will also be hierarchical. This means that if a command is run in hierarchical mode on a high-level element (for example, a project), it can take several minutes to run, as the command will be executed on all the components.

The module allows several users to carry out check-out operations on units defined in the same package, without having to run a check-out on the package itself. In a situation like this, it is recommended that you avoid running a check-out on the package itself in non-hierarchical mode, so as to allow the possible creation of new units.

When the "Check-in", "Import from repository" and "Check-out" commands are run, a confirmation dialog box (Figure 2-1) is displayed (only if the "Confirm operations" tick box has been checked during module configuration, through which it is also possible to define whether or not the command is to be run in hierarchical mode or not).

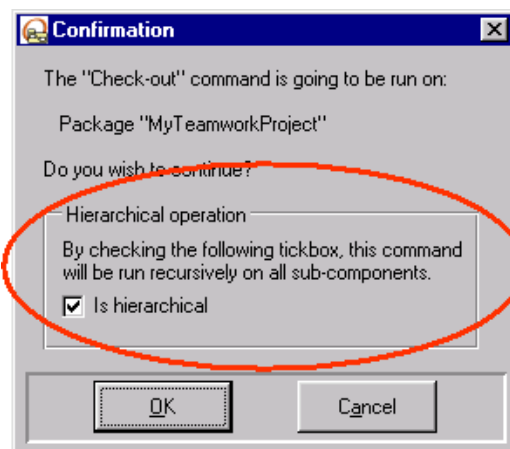


Figure 2-1. The hierarchical mode confirmation dialog box

Summary of commands available in read-only mode

| The ... command | is used to ... |
|------------------------|--|
| Check-out | reserve an element, in order to subsequently modify it. The latest version of the element in question is imported into the Objecteering/UML model. |
| Import from repository | import the complete element stored in the repository into the Objecteering/UML modeling project. |

Summary of commands available in read-write mode

| The ... command | is used to ... |
|------------------------|---|
| Check-in | free the element and export it into the repository, so that the element is available to other users in its latest version. |
| Check-in and check-out | check-in and then check-out model elements. By doing this, model elements are updated for all other users, but remain locked against further modifications. |
| Undo check-out | cancel the check-out and go back to the previous version. |

Summary of commands available on all elements (read-only and read-write)

| The ... command | is used to ... |
|------------------|--|
| Project check-in | check-in all project elements currently in checked-out state. This command cannot be run on the current element, but only the entire Objecteering/UML project. |
| Properties | display the element's properties. |

Unselecting the module

When the module is unselected, a check is run to ensure that all elements in the current UML modeling project are in read-only mode, which signifies that these elements have been archived.

If elements have remained in read-write mode, a confirmation dialog box allows you to force unselection (Figure 2-2).

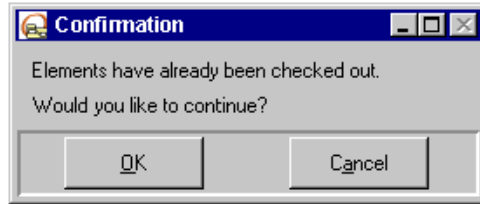


Figure 2-2. Confirming module unselection

The module unselection operation puts all elements into read-write mode.

Commands available on elements in read-only mode

Overview

Elements in read-only mode cannot be directly modified. The available commands are used to make elements modifiable or to import them into the repository as they are.

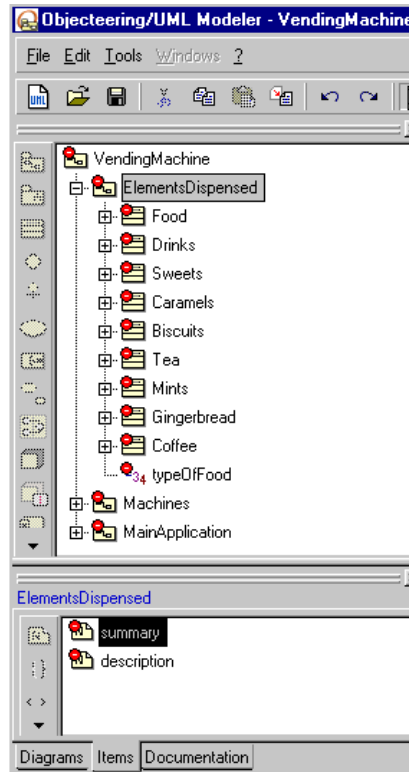


Figure 2-3. Representation of elements in read-only mode

Check-out

This command allows you to mark one or more units as reserved. Thus, if another user tries to make a reservation at a later date, the command will not work. When a check-out is run, the model is first updated, in order to carry out the modifications only on the latest version of the unit.

In hierarchical mode, the components of the element are also checked-out and imported.

Import from repository

This command updates the element with the contents present in the repository. It is used to retrieve all the modifications made to an element, without actually checking out the element in question.

Commands available on elements in read-write mode

Overview

Elements in read-write mode can be directly modified. The available commands are used to make these elements non-modifiable, by integrating the modifications into the repository ("*Check-in*") or by cancelling the modifications carried out ("*Undo check-out*").

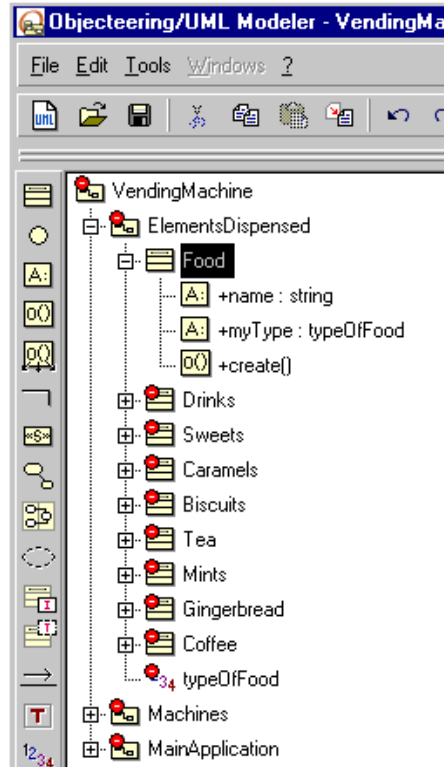


Figure 2-4. Representation of elements in read-write mode

Check-in

This command is used to check-in and export the unit which was previously reserved or which has just been created by the user.

At the end of this operation, those elements on which the command has been run are unlocked and can be reserved by another user.

This command can only be run on components which have been checked-out.

When the "*Check-in*" command is run on the elements concerned, they are subsequently put into read-only mode.

In certain cases, units which depend on other units, on which a check-out is requested, can be automatically checked-in, in order to make the project consistent with the repository. For example, a new unit cannot be checked-in, in order to retain the composition between units in the multi-user repository.

If the element has not been modified, then an "*Undo check-out*" is automatically run, instead of a check-in. This means that the element's version is not incremented if the element is identical to the previous version.

For the *Objecteering/ClearCase* and *Objecteering/SCC* modules, a comment can be entered on the check-in. A dialog box is displayed; if the "Confirm operations" module parameter has been activated and the element has been modified (if this is not the case, another check-out is carried out and no comment is requested).

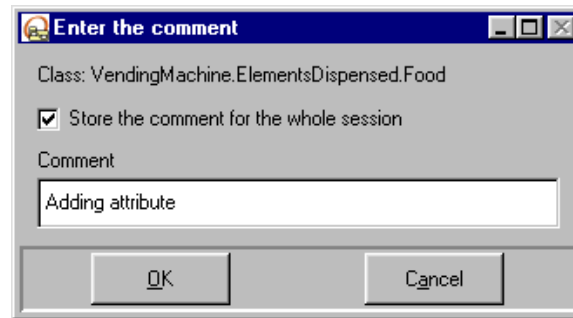


Figure 2-5. Comment entry dialog box for a check-in

The "Store the comment for the whole session" tickbox allows you to apply the same comment or a different comment to all the components. Where the check-in is carried out hierarchically, the comment entered the time before is then re-used.

Note: Clicking on the "Cancel" button cancels the check-in for the current unit and all the following units.

Check-in and check-out

This command is used to check-in and export a unit into the repository, thereby making it available to other users working on the multi-user project, and then to check-out this unit once again, in order to continue working on it.

This command can only be run on checked-out components.

For further information, please refer to the "*Check-in*" and "*Check-out*" commands.

Undo check-out

This command is used to undo the check-out of the element and abandon the reservation. The repository is not updated, but elements for which the check-out has been cancelled are re-imported into the Objectteering/UML modeling project, in order to retrieve their latest version.

At the end of this operation, those elements on which the command has been run are unlocked and can be reserved by another user.

This command can only be run on checked-out components.

Commands available on all elements (read-only and read-write)

Project check-in

This command is used to carry out a check-in of all the elements of the project which have been checked-out by the user in a single operation.

If the "*Confirm operations*" tick box has been checked at module configuration level, confirmation is requested before the check-in of all the elements is run.

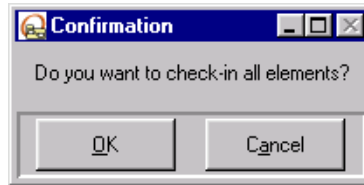


Figure 2-6. Confirmation dialog box in the "*Project check-in*" command

Properties

This command is used to display information on the current element. The behavior of the command varies depending on which module is being used.

For further information, please refer to the chapter related to the module in question in this user guide (chapters 6, 7 and 8).

Chapter 3: Typical usage

Overview of typical teamwork module usage

Introduction

The aim of this chapter is to provide you, the user, with further, complementary information on the configuration and implementation of the Objectteering/UML teamwork modules *Objectteering/Multi-user*, *Objectteering/SCC* and *Objectteering/ClearCase*.

In this section, you will find information on the following areas:

- ◆ creating an initial model before sharing it with other users
- ◆ creating, deleting, renaming and moving multi-user atomic units
- ◆ module behavior with regard to links between elements
- ◆ what to do after a problem occurs with Objectteering/UML
- ◆ managing work products and generated files

The advice and tips given in this chapter are the result of the vast experience gained in industrial development using Objectteering/UML by development teams of all different sizes.

Defining the initial model

Introduction

Before starting work on defining the initial model, each user participating in the teamwork project must have a database (locally on his machine, if possible), and a shared directory must already exist. Access is possible via UNC paths (\\machine, etc).

The multi-user directory is common to all users wishing to work on the same model. If several models are shared independently, you should:

- ◆ create different multi-user directories
- ◆ allocate the correct value at module configuration level for the Objectteering/UML modeling projects concerned

The name of the multi-user directory directly corresponds to the physical location of the directories created for the storage of models.

Users who wish to be able to modify the model must be able to access these directories in read-write mode.

The UML modeling project start-up phase is important. The initial structure of the model is used to:

- 1 - define the system's principal packages and the dependencies between these packages
- 2 - organize the work to be carried out by the first people to work on the UML modeling project

During this phase, changes made to the structure of the model can be important and non-optimal if special care is not taken. We recommend that you:

- 1 - Define the principal packages collectively, in order to create an initial model which corresponds to a perception common amongst the different players. Typically, it is possible to start with "*blackboard*" definition meetings, whose aim is to identify the system's principal packages, the choice of UML models and the functional division of the application in terms of packages and actors.
- 2 - Build an initial work plan with the affectation of packages to different players.
- 3 - Define consolidation meetings, whose purpose is to collectively review the structure of the model, where necessary.

At Objecteering/UML level, this principle is represented by:

- ◆ the constitution of the initial model from a first Objecteering/UML modeling project, followed by the initial check-in of the model (this is automatically carried out when the module is selected)
- ◆ the creation of a database for each user
- ◆ the reservation (check-out) and delivery (check-in) of each person's work
- ◆ the taking into account of other users' work (Import...)
- ◆ if the structure is questioned, we recommend that you:
 - 1 - Integrate into a UML modeling project all the latest modifications (Import)
 - 2 - Reserve impacted elements (possibly the entire project) through a check-out
 - 3 - realize and deliver structural modifications (check-in)
- ◆ synchronize all users from the new structure through a "*Import complete element from repository*" on the entire project

Creating a "model" database

To reduce the administration work previously described, we recommend that you prepare a "model" database, which will be used in the creation of different users' private work spaces. This database should be initialized according to the sequence described above, except for the activation of the *Objecteering/Multi-user* module.

Private work spaces can then be created by copying the "*model*" database, using the Objecteering/UML database administration tool. Only the activation of the *Objecteering/Multi-user* module is necessary for each individual private work space.

Multi-user atomic units

Adding elements

When adding elements, you must carry out a check-out on the embedding element.

To save the new element, the user should simply carry out a check-in on the embedding element. The check-in on the new element may be directly accepted, but an *"Undo check-out"* without import on the parent is forbidden.

Deleting elements

To delete model elements, the standard Objectteering/UML delete command should be used.

Warning!

This deletion can only be run:

- ◆ on an element which has been reserved through a check-out
- ◆ on an element whose embedding element has been reserved through a check-out

Destroying a referenced element

With regard to element references, the behavior of the teamwork module being used differs from that of the explorer. We therefore recommend against the use of module commands on references. The sequences below should be respected according to their context.

To delete a referenced element, both the element itself and its parent must be in read-write mode.

- 1 - First run the check-out command on the package to which a reference is associated.
- 2 - Delete the referenced element using the classic Objectteering/UML *"Delete"* command (available through the *"Edit/Delete"* menu or via the *"Delete"* keyboard key).
- 3 - Run the check-in command on the package to which a reference was associated.

Renaming elements

The renaming of elements is an operation which is carried out on reserved elements (elements in check-out).

After a renaming operation, we recommend that all those involved carry out an *"Import element using current mode"* on a package encompassing all impacted elements, on the first level package or on the entire UML modeling project, in order to update all the local model's links.

When *"circular"* renaming is to be carried out (for example, if the user has two classes, C1 and C2, and he wishes to change the names of these classes from C1 to C2 and from C2 to C1), the following operations should be carried out:

- ◆ Check-out C1 and C2.
- ◆ Change the name of C2 to xC2.
- ◆ Change the name of C1 to C2.
- ◆ Change the name of xC2 to C1.
- ◆ Check-in C2 and xC2.

Moving elements

When an element is moved, the embedding origin and destination element must be reserved. To retain references to the element which is moved, the drag & drop operation should be used, and not the cut/paste or the copy/paste operations.

After moving an element, we recommend that all those involved carry out an *"Import element using current mode"* on a package encompassing all impacted elements, on the first level package or on the entire UML modeling project, in order to update all the local model's links.

Adding and modifying diagrams

Bearing in mind that diagrams are associated with model elements, the element in question must be reserved (check-out) before the creation or modification of any diagram.

Links

Introduction

Two types of link between modeling elements can be distinguished:

- ◆ links which necessitate the prior check-out of all linked elements (this is the case for non-oriented links)
- ◆ links which only necessitate the check-out of the origin elements (this is the case for oriented links)

Adding non-oriented links

In the case of non-oriented links, all linked elements must be checked out before the creation of the link within the model.

The types of link concerned are:

- ◆ associations and aggregations between classes, nodes, components, objects, etc.
- ◆ communication links between actors and use cases
- ◆ DataFlows between packages, classes, ...

Note: If the user wishes to add an association between two classes belonging to two different packages, then he must carry out a check-out:

- ◆ either on both the classes
- ◆ or on the package for one of the classes and then on the class belonging to the other package
- ◆ or on both the packages

A typical example of this is where work is carried out on a package, where it is then necessary to add a link towards a class belonging to another package.

Adding oriented links

Where oriented links are concerned, only a check-out of the origin element is required.

The types of link concerned are:

- ◆ generalizations between packages, classes, actors, use cases, nodes, etc.
- ◆ dependencies between packages, classes, actors, components, etc.
- ◆ implementation links between classes, components and interfaces

For this type of link, other than the check-out of the origin element, no particular precautions are necessary.

Deleting links between elements

The same recommendations must be taken into account when links between elements are deleted as when they are added.

For oriented links, a check-out of the origin element must be carried out before deletion.

For non-oriented links, a check-out of the origin and destination elements must be carried out before deletion.

Link circularity problems

After “*concurrent*” model modifications, it is possible to produce circular dependencies between packages.

Example:

On a model which has two packages, P1 and P2, a user U1 carries out a check-out on P1, adds a dependency from P1 to P2 and then carries out a check-in. During this time, a second user, U2, is carrying out the opposite operations. The result of these modifications renders importing an update impossible, since Objectteering/UML displays an error message, due to the mutual dependency between packages.

To correct this type of problem, one of the users should simply re-execute the check-out of his package and replace one of the circular dependencies by a reference link.

Reference links are not graphically visible, but can be entered using the icons situated in the central column of the explorer. They are used by Objectteering/UML to calculate visibility between modeling elements, but are not checked for circularity.

Once a model containing all the modifications has been built, we recommend that the user review this model, in order to check that the circularity (now with referencing) is not linked to the incorrect structuring of the applications.

How should I proceed after a problem?

Objecteering/UML crashes

- 1 - Remove the locks.
- 2 - Run a synchronization if the command was a check-in, a check-out or an undo check-out.
- 3 - Relaunch the command for all elements on which it failed.

Consistency checks

If a model cannot be imported because of consistency checks, you should carry out the following operations:

- 1 - Deactivate consistency checks.
- 2 - Check-out the part of the model causing consistency errors.
- 3 - Correct the model.
- 4 - Activate consistency checks.
- 5 - If consistency checks have been activated, check-in the modified elements.

How can I resolve a problem of dependency cycles?

If two users create a dependency cycle between two packages and validate their actions via the check-in command, then no check-out commands can be carried out on "*destination*" packages.

To solve this problem, re-run a check-out of the "*origin*" packages, change the use link into a referencing link and then confirm these changes by carrying out a check-in command.

Managing work products and generated files

Taking into account modifications in generated files

The *Objectteering/Multi-user*, *Objectteering/ClearCase* and *Objectteering/SCC* modules do not manage generated files and consider these to be independent files.

In *Objectteering/UML*, there are three ways of modifying the code of a generated file:

- 1 - Modification of notes (C++, Java, etc.)
- 2 - Use of the module external edition command
- 3 - Modification of generated files outside *Objectteering/UML*

As regards this third manner of modifying generated file code, a command available for each code generation module must be run, allowing all modifications carried out directly in generated files to be reported to *Objectteering/UML*.

This command can only be carried out if the modified elements have been checked-out. If a check-in is run on elements without having run this command, all modifications carried out in generated files will be lost.

The *Objectteering/Multi-user*, *Objectteering/ClearCase* and *Objectteering/SCC* modules all provide the "*Update generation work products before check-in*" module parameter, used to automatically carry out this code retrieval before running a check-in.

Generating C++ or Java sources in personal environments

In a UML modeling project managed in multi-user mode, work products are common to all users and sources are generated in the same directory which is common to all users. In a Windows environment, it is possible to define an identical path for all users, but which is specific to each individual user (for example, "C:\Project\sources"). In this way, even though you have an identical path to all users, each user generates in his own environment.

To solve this problem in Unix, and to define a different generation path for each user (even in Windows), the "*Update generation work products after import*" module parameter is used to modify each work product after the "*Import from repository*" command or after a check-out, so that the parameterization of this product be proper to the user.

By activating this parameter, generation paths defined for each work product are updated with the default value (that defined by the parameterization of the C++ or Java modules).

Note: This update only concerns C++ and Java work products, for modules delivered as standards and for user modules inheriting from these two modules.

Chapter 4: Transferring elements between UML modeling projects

Running principle

Description of the transfer function

The transferring elements feature (shown in Figure 4-1) gives the user the possibility of importing elements which come from another UML modeling project into his current UML modeling project. The elements imported can be packages or classes.

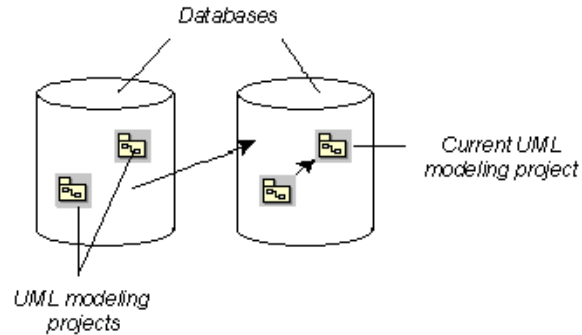


Figure 4-1. The transfer function allows the import of elements which come from other UML modeling projects

Note: If the elements which are to be imported come from another database and do not figure in an Objectteering/UML modeling project, then they must not already be present in the repository. However, if the elements already exist in the Objectteering/UML modeling project, then they must also already exist in the repository. They are updated in Objectteering/UML when the *"Import from repository"* command is run, and updated in the repository when the check-in command is run.

To import elements from other Objectteering/UML databases, the following steps should be carried out:

- 1 - Carry out a check-out on the UML model root in *"non-hierarchical"* mode.
- 2 - Import the package through the classic import window.
- 3 - Finally, carry out a check-in on the UML model root in *"hierarchical"* mode.

Identifying model elements

Objectteering/UML identifies all elements entered graphically (diagrams) or in the model explorer, in a universally unique way (*site/base/project/element*) as soon as they are created. Transfer logic uses these identifiers to ensure their automatic update.

Example:

- 1 - Create the C1 class in the P1 UML modeling project.
- 2 - Transfer C1 to the P2 UML modeling project.
- 3 - Rename C1 as C2 in the P2 UML modeling project.
- 4 - Transfer C2 from the P2 UML modeling project to the P1 UML modeling project.

Result: C2 replaces C1 in the P1 UML modeling project model.

Import logic

Import operations use identifiers in their logic.

- 1 - A copy of the element is made, if elements with the same identifier do not yet exist.
- 2 - If an element with the same identifier already exists, it is replaced.
- 3 - If a newly-imported element has links towards other elements, the links are retained if the linked elements with the same identifiers exist. If not, the links are destroyed.

For example, the import of the C1 class into an empty UML modeling project creates this class, but destroys its associations, generalizations, and so on. The import of the S1 diagram into an empty UML modeling project creates the diagram and its classes and creates the links between these classes, but not those which reference external elements.

Note: For further details on import operations, please refer to the "*Importing elements between UML modeling projects*" section in the current chapter of this user guide.

Importing elements between UML modeling projects

The "Import" dialog box

The "Import" dialog box (shown in Figure 4-2) contains a list of available UML modeling projects, and details of their contents, presented in the form of a hierarchical explorer.

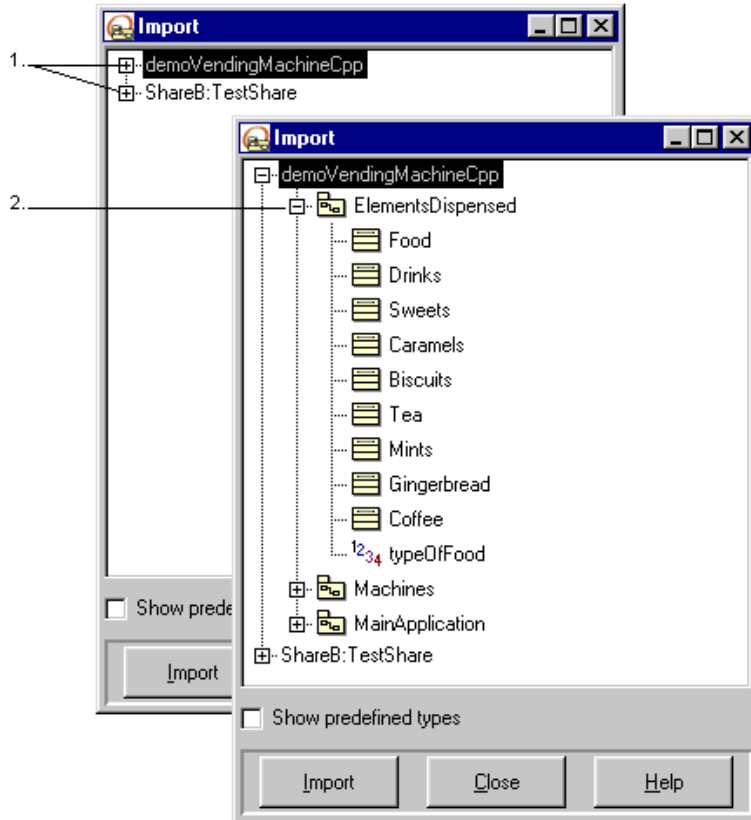


Figure 4-2. The "Import" window and its components

Chapter 4: Transferring elements between UML modeling projects

Description:

- 1 - UML modeling projects.
- 2 - Components of the selected UML modeling project. To display the components of a given UML modeling project, simply click on the + boxes in the tree structure.

Imported objects

Component elements, which appear when you click on the "+" boxes in the "Import" window tree structure, are listed in the following order: packages and classes. The import will fail if there is any inconsistency between imported elements.

| Object ... | imported component elements... |
|----------------------|---|
| UML modeling project | the whole UML modeling project (packages, classes, ...) |
| Package | classes (with their operations, attributes, "visible" associations), documents, tagged values, diagrams |
| Class | operations, attributes, "visible" associations, documents, tagged values, diagrams |

Non-imported objects are:

- ◆ reference links from a package to another element which is not imported and which does not already exist in the current UML modeling project
- ◆ non-oriented associations (visibility NULL on both sides)

Note 1: Before importing an element, the check-out command must be run on the UML model root, in non-hierarchical mode.

Example of an import

Figure 4-3 describes how to carry out an import from a package into the current UML modeling project.

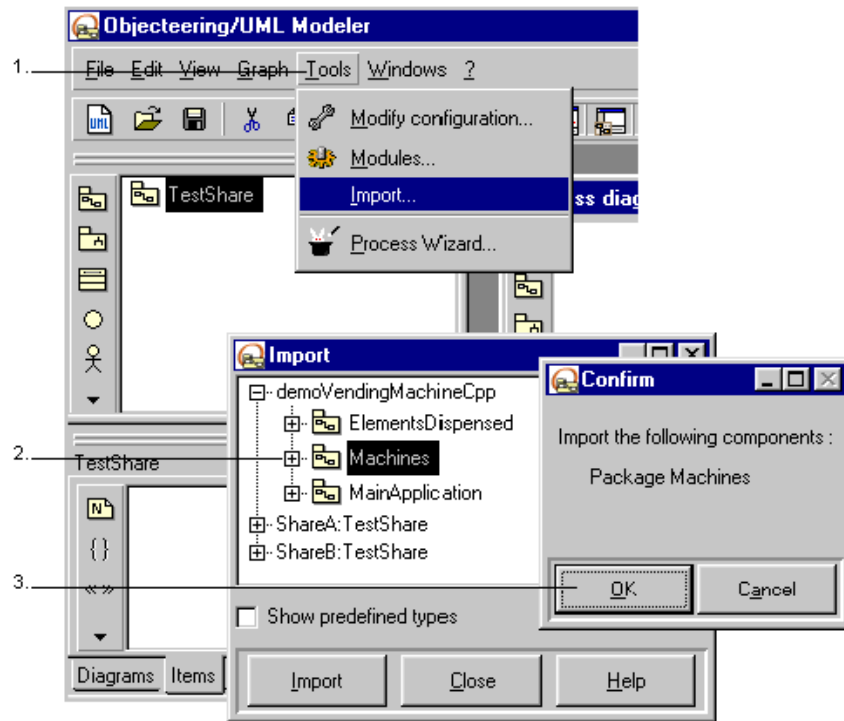


Figure 4-3. Example of an import of a package into a UML modeling project

Steps:

- 1 - Click on the "*Tools/Import*" menu.
- 2 - Select the package you wish to import.
- 3 - A confirmation dialog box then appears, indicating the element you have selected for import. Click on the "OK" button to confirm.

The selected package is then imported into your current UML modeling project. The "*Import*" window remains open, in case you wish to import other elements. If this is not the case, simply close the window.

Note: The selection of a UML modeling project from the list of components replaces the current UML modeling project by the selected UML modeling project. It deletes those elements in the current UML modeling project which do not exist in the selected UML modeling project, replaces those which exist in the two UML modeling projects and adds those which did not already exist in the current UML modeling project.

Chapter 5: Parameterizing the
Objecteering/UML
teamwork modules


Defining module parameters

Introduction

When a teamwork module is selected, module parameters are automatically initialized with optimal default values.

This section presents the different module parameters, and describes their impact on the behavior of the teamwork modules.

Configuring the Objecteering/UML teamwork modules

The Objecteering/UML teamwork modules can be parameterized through the "*Modifying configuration*" dialog box (as shown in Figure 5-1 below), which is launched by clicking on the  "*Modify module parameter configuration*" icon.

The "General settings" parameter set

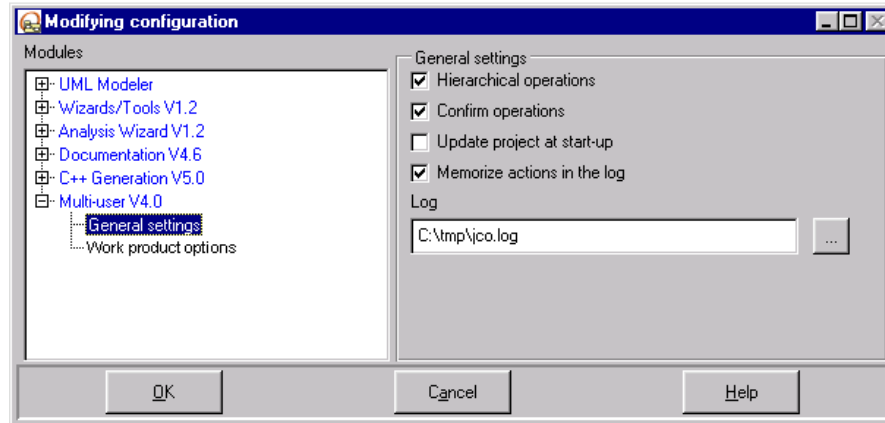


Figure 5-1. The "General settings" set of parameters for the *Objecteering/Multi-user* module

| The ... parameter | is used to ... |
|-----------------------------|---|
| Hierarchical operations | run commands on components. This parameter determine the hierarchical or non-hierarchical functioning modes of the check-out, check-in and model import operations (to know whether, for example, a package check-in automatically leads to the check-in of the classes it contains). |
| Confirm operations | select a mode employed by users to confirm operations. If this option is activated, certain commands run by the user will be subject to interactive confirmation. |
| Update project at start-up | automatically run the "Import from repository" command when Objectteering/UML is started. By doing this, the project is automatically updated with regard to the repository. Those elements which are still in check-out are not updated. |
| Memorize actions in the log | record in a file all actions carried out by the user. By default, this file is created in the repository directory, and its name corresponds to the user's name followed by the ".log" suffix. |
| Log | define the file which will contain a record of all the actions carried out by the module. This parameter is only used if the " <i>Memorize actions in the log</i> " parameter has been activated. |

Note: The parameters described above exist for the *Objectteering/SCC* and *Objectteering/ClearCase* modules, as well as the *Objectteering/Multi-user* module.

The "Work product options" parameter set

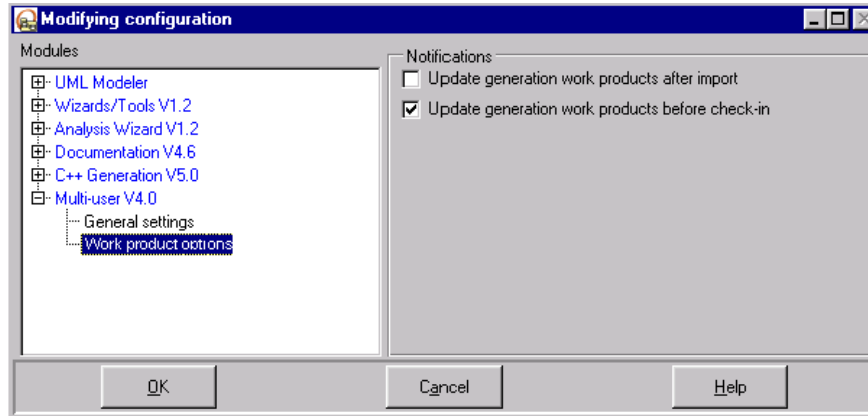



Figure 5-2. The "Work product options" set of parameters for the *Objecteering/Multi-user* module

| The ... parameter | is used to ... |
|---|---|
| Update generation work products after import | update the paths of C++ generation work products (code generation and makefiles) and Java generation work products, using the values defined during C++ or Java module parameterization. |
| Update generation work products before check-in | reverse the contents of the files generated before carrying out the check-in. This parameter is indispensable if the source code is modified outside Objecteering/UML. If this is the case, this parameter allows modifications made to the source code to be automatically incorporated, before carrying out the check-in operation. |

Note: The parameters described above exist for the *Objecteering/SCC* and *Objecteering/ClearCase* modules, as well as the *Objecteering/Multi-user* module.

Graphic representation of the read-only and read-write modes

Overview

By default, *Objectteering/UML Modeler* graphically represents elements in read-only mode through the  "No entry" icon superimposed over the icon representing the model element itself. Elements in read-write mode are simply represented by their icon with no additional mask superimposed.

The user can, however, choose to implement a bitmap indicating the read-write status of modeling elements. To do this, he should simply change the name of the "RWmode_optional.gif" file, which is delivered as standard in the \$OBJING_PATH/res directory, to "RWmode.gif".

The user can use the bitmap of his choice to represent the read-write state of model elements.

It is also possible to change the read-only mode bitmap, by modifying the "ROmode.gif" file in the \$OBJING_PATH/res directory.

Example of graphic representation of the read-only and read-write modes

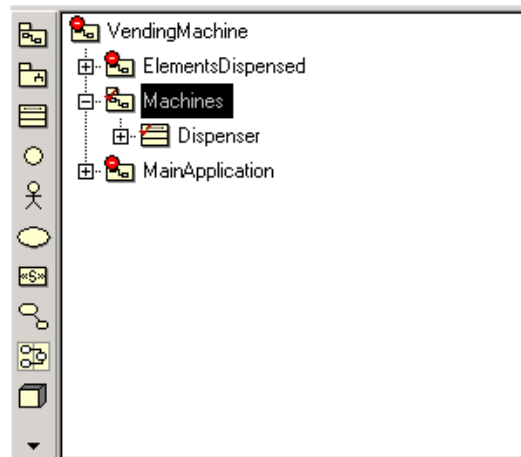


Figure 5-3. Graphic representation of the read-only and read-write modes

The administration module

Overview

The administration module should be used when module commands are not functioning, when the user is "*blocked*" or when Objecteering/UML has crashed and consistency must be re-established between the repository and the UML modeling project.

The aim of this module is to allow the user to unblock a Objecteering/UML modeling project which is in an incoherent state.

Warning! It is imperative that you have a sound knowledge of the workings of the module, in order to be able to use this administration module with no risk of losing valuable data.

The administration module must not be installed by default in user databases, but only if the teamwork module functions incorrectly. When it is being used, it is preferable that no other users use the teamwork module (check-in, check-out or import commands).

All commands are available on elements in both read-only mode and read-write mode.

If the "*Hierarchical operations*" tickbox has been checked in the module configuration window, then the effect of these commands is propagated on all sub-elements.

Note: The module must have already been selected in order for the administration module commands to work properly.

Administration module commands

| The ... command | is used to ... |
|--------------------------------------|---|
| Generate element in repository | generate a multi-user atomic unit in the form of a file in the repository. |
| Modify the parameters | modify module parameters, which should be modified with care. |
| Synchronize element state in project | synchronize the state of a multi-user atomic unit with regard to its state in the repository. |
| Unlock repository | delete the locks present on the repository. |

The "Generate element in repository" command

The "*Generate element in repository*" command can be run on all multi-user atomic units and is used to generate a multi-user atomic unit in the form of a file.

When the "*Confirm operations*" module parameter has been activated, confirmation is requested before the command is launched. The confirmation dialog box which appears allows you to specify the directory where the element is to be externalized, as well as indicating whether the command should be run in hierarchical mode or not (as shown in Figure 5-4).

Note: It is not necessary to create the externalization directory before running the command. This directory will be automatically created by the command.

Two possibilities are available to the user:

- 1 - Generation directly in the repository. By default, the repository associated with the Objecteering/UML is proposed.
- 2 - Generation in a directory other than the repository.

If the element is generated directly in the repository, the user must make sure that he has the latest version of the element before launching the command, as other users will retrieve the version which is going to be generated in the repository.

Chapter 5: Parameterizing the Objectteering/UML teamwork modules

For the *Objectteering/ClearCase* module, if the element is generated directly in the view, the file corresponding to the element must first be checked-out from the ClearCase interface and in the Objectteering/UML modeling project, and then a check-in run on this file before launching the "Generate element in repository" command.

For the *Objectteering/SCC* module, the file generated is not transferred to the provider, but is instead exclusively generated in the directory defined by the "Work directory" module parameter. In this case, the retrieval operation should be run in the provider.

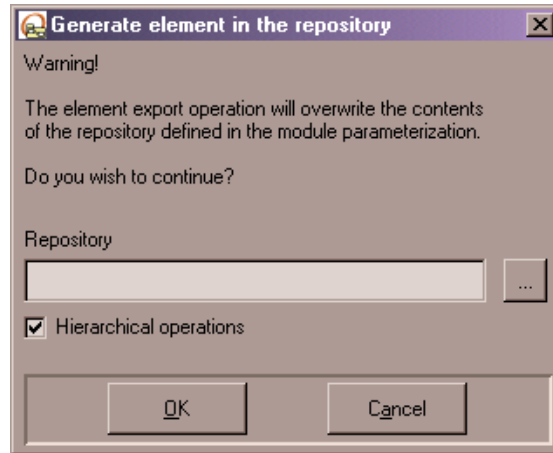


Figure 5-4. Running the "Generate element in repository" command

The "Modify the parameters" command

The "*Modify the parameters*" command is used to modify the module's sensitive parameters.

For the *Objectteering/Multi-user* module, this command is used to modify:

- 1 - The name of the user. All the users connected to the same repository must have different user names. If this parameter is modified, we recommend that you first run a "*Project check-in*" operation, to check-in all those elements which have been checked-out. If the "*Modify the parameters*" command is run while elements are still checked-out, the state of the elements in the Objectteering/UML model will be out of synch with regard to the repository, and the modifications made to these elements could be lost.
- 2 - The directory where the repository is stored. After modifying this parameter, we recommend that you run the "*Import from repository*" command on the UML modeling project, in order to obtain the latest version of the model stored in the new repository. If this new repository is completely different from the former one, you should create a new UML modeling project, install the teamwork module and connect to the new repository.
- 3 - The optimization mode. For reasons of performance, we recommend that you never deactivate this parameter. If this parameter is deactivated, the version of elements will not be used to optimize "*Check-in*", "*Check-out*" and "*Import from repository*" operations. If this parameter is active, all elements will be imported into Objectteering/UML, even if no modifications have been made to them in the repository.

Given the importance of these parameters, confirmation is systematically requested (with warnings).

The "*Modify the parameters*" command is not available in the *Objectteering/SCC Administration* module.

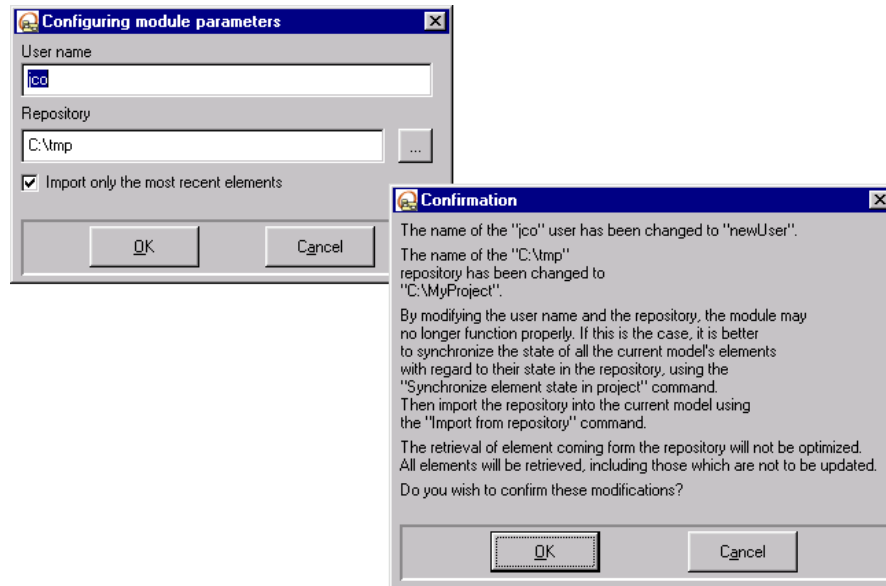


Figure 5-5. Running the "Modify the parameters" command on the *Objecteering/MultiUser* module

The "Synchronize element state in project" command

The "*Synchronize element state in project*" command synchronizes the state of a multi-user atomic unit with regard to its state in the repository. This command does not re-import elements, but simply updates their check-in/check-out state in Objecteering/UML. This command must only be used when a standard operation cannot be carried out, when Objecteering/UML and the repository are out of synch.

A save is made if at least one element has been resynchronized.

When the "*Synchronize element state in project*" command is run, its result can be seen in the Objecteering/UML console.

The "Unlock repository" command

The "*Unlock repository*" command is used to unlock the repository after a system error.

For the *Objectteering/Multi-user* module, this command deletes:

- ◆ all the files present in the "Locks" directory with the ".read", ".write" and ".cout" extensions
- ◆ the "multiuser.lock" file

For the *Objectteering/ClearCase* module, this command runs:

- ◆ an "Undo check-out" of the file associated with the project located in the "Project" directory if this file is checked-out
- ◆ an "Undo check-out" of the <username>.read file located in the "Locks" directory, if this file has been checked-out by the user who is running the command
- ◆ a "Check-in" of the "Actor", "Class", "Component", "DataType", "Item", "Locks", "Package", "Project", "Node", "Signal" and "UseCase" directories

For the *Objectteering/SCC* module, this command runs:

- ◆ an "Undo check-out" of the file associated with the project located in the "Project" directory if this file is checked-out
- ◆ a "Check-in" of the users.txt file, if this file has been checked-out by the user who is running the command and if it has been modified, or an "Undo check-out" if it has not been modified
- ◆ an "Undo check-out" of the <username>.read file located in the "Locks" directory if this file has been checked-out by the user who is running the command

For further information on repository locks, please refer to chapter 6, 7 or 8 of this user guide, depending on the *Objectteering/UML* teamwork module you are using.

Chapter 6: The Objecteering/Multi-user
module

Introduction to the Objecteering/Multi-user module

Functions

Using the *Objecteering/MultiUser* module, it is possible to:

- ◆ Check-in and/or check-out a model element
- ◆ Undo check-out
- ◆ Update the Objecteering/UML model

Saving/Restoring

The use of the *Objecteering/Multi-user* module does not replace the action of backing up directories and files. In the case of the restoration of files following a system problem, you should be careful regarding ".lock" files present in the root directory after restoration. It is, therefore, imperative that:

- ◆ all users unselect the *Multi-user* module and quit the editing of their model
- ◆ all ".lock", ".cout", ".read" and ".write" files be destroyed after the restoration of the files

All users can then restart work on the restored version, by selecting the *Multi-user* module and defining the user name and the multi-user directory.

Using the Objectteering/Multi-user module

Initializing the repository

To initialize the repository, the following sequence of operations must be carried out:

- 1 - A UML modeling project must be created. First, define a name and a location for this new UML modeling project in the first two fields of the "Create a UML modeling project" dialog box below. Next, check the "Model root name" tickbox and define the model name in the associated field (as shown in Figure 6-1). This model name is important for what will follow, as all users have to define the same model root name when creating their UML modeling projects, so as to be able to access the associated repository.

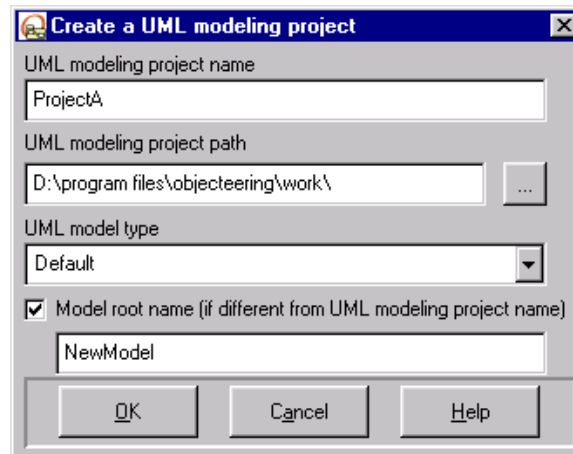


Figure 6-1. Creating a UML modeling project

- 2 - The main elements of the UML modeling project must be created. We recommend that you create an initial model structure, within which other users can start their modeling work. Without this initial structure, users have no predefined model structure and cannot start modeling in an organized way.

- 3 - The *Objecteering/Multi-user* module must be selected for the new UML modeling project. To do this, run the "Modules" command from the "Tools" menu, and define the current user name and repository path (as shown in Figure 6-2).

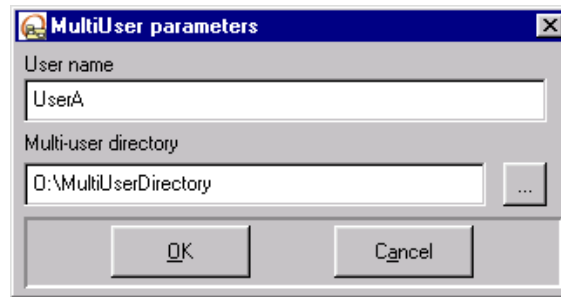



Figure 6-2. Defining Multi-user parameters

Chapter 6: The Objecteering/Multi-user module

After the *Objecteering/Multi-user* module has been selected:

- ◆ A directory hierarchy is automatically created in the repository. The initial model elements are positioned there, in the form of externalized ASCII files.
- ◆ The  icon appears on all model elements in the Objecteering/UML explorer (as shown below in Figure 6-3).

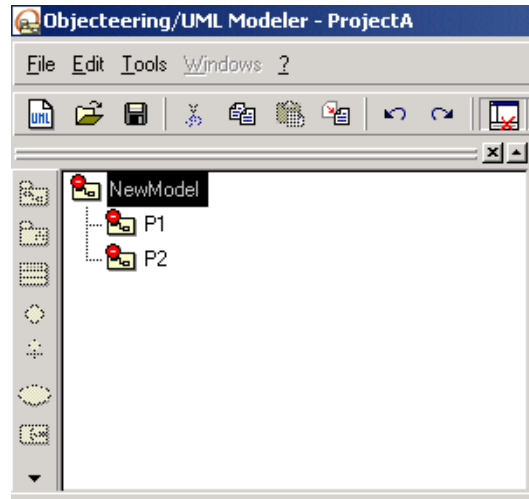


Figure 6-3. The explorer after selection of the *Objecteering/Multi-user* module

Initializing private work spaces

Once the repository has been created, each user must initialize his own private work space, by carrying out the steps below.

- 1 - A new Objecteering/UML modeling project must be created. First, define a name and a creation location for this new UML modeling project in the first two fields of the dialog box shown below. Next, check the "Model root name" tickbox and enter in the associated field **exactly the same name** as that used when creating the repository (as shown in Figure 6-4).

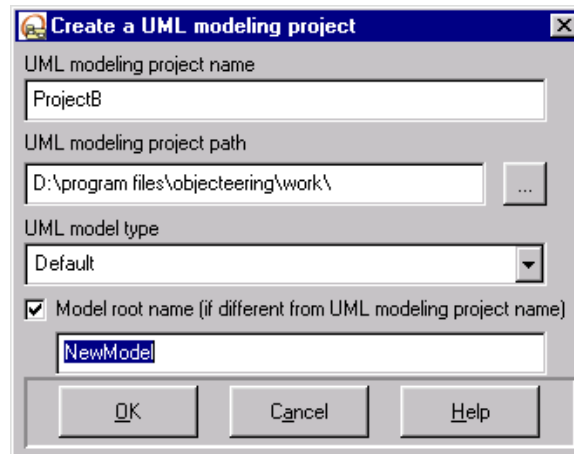


Figure 6-4. Creating a user's private work space

- 2 - The *Objecteering/Multi-user* module must be selected for this new UML modeling project. To do this, run the "Modules" command in the "Tools" menu, and allocate the current user name and the path to the existing repository (Figure 6-5).

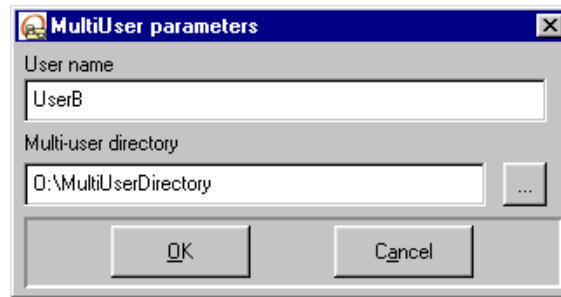


Figure 6-5. Defining Multi-user parameters

If the UML modeling project is not empty, all elements present therein will be deleted. Confirmation is requested before retrieving elements present in the repository (as shown in Figure 6-6).

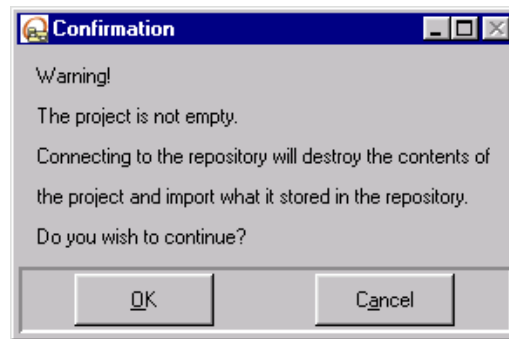



Figure 6-6. Confirmation dialog box before retrieving the model from the repository

After activating the *Objecteering/Multi-user* module:

- ◆ the data present in the repository is automatically imported into the user's private work space, and the model reconstructed
- ◆ the  icon appears on all the model elements in the Objecteering/UML explorer (as shown in Figure 6-7 below).

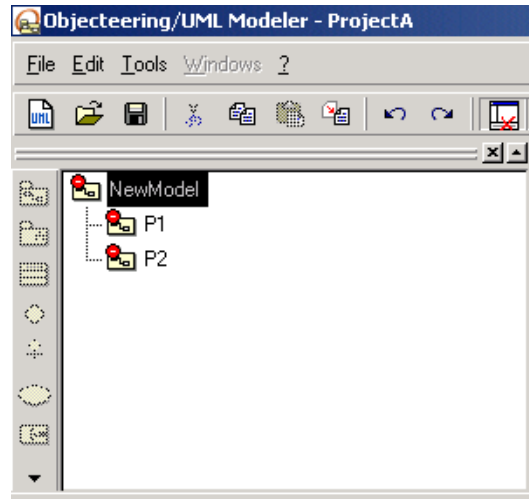


Figure 6-7. The explorer after selection of the *Objecteering/Multi-user* module

Objecteering/Multi-user commands

Overview

The following commands are specific to the *Objecteering/Multi-user* module.

| The ... command | is used to ... |
|-----------------|---|
| Get information | list new elements, reserved or not yet reserved in the UML modeling project, with regard to the repository. |
| Properties | display an element's properties. |

The "Get information" command

The "*Get information*" command allows you to list the following items in the Objecteering/UML console.

- ◆ elements which are reserved by the current user or by another user
- ◆ newly created elements or elements which have not yet been recorded in the repository
- ◆ elements which are not in the UML modeling project
- ◆ elements which are present in the model but destroyed in the repository

This command is always carried out in hierarchical mode, so as to handle all the components of the current element. When this command is run on the UML modeling project, all information on all project elements is obtained.

The "Properties" command

For the *Objecteering/Multi-user* module, the information displayed is as follows:

- ◆ The location of the repository
- ◆ The name of the file corresponding to the element
- ◆ The version of the element present in the Objecteering/UML model
- ◆ The version of the element present in the repository
- ◆ The state of the element (checked-in, checked-out, new, deleted)

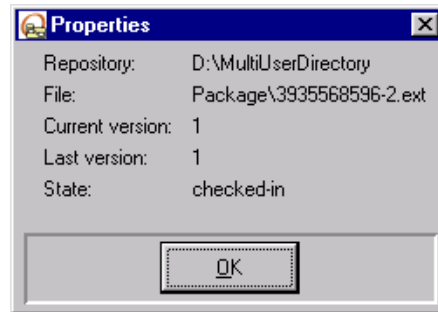


Figure 6-8. The properties of an element with the *Objecteering/Multi-user* module

Chapter 7: The
Objecteering/ClearCase
module

Introduction to the Objecteering/ClearCase module

Overview

The *Objecteering/ClearCase* module has been designed to function with *ClearCase 4.0*, *ClearCase 4.1*, *ClearCase 4.2* and *ClearCase 5.0*.

The *Objecteering/ClearCase* module is used to configure elements of an Objecteering/UML model, to carry out the most frequently used ClearCase operations and to simplify a developer's work in the domain of configuration management.

The functions of the *Objecteering/ClearCase* module, necessary to configuration management, are accessed through the Objecteering/UML interface.

The *Objecteering/ClearCase* module does not substitute the ClearCase tool. The administration of this tool is not, therefore, available from Objecteering/UML. Notably, it is necessary for ClearCase to be active and for a view to have been selected before launching Objecteering/UML.

Limitations

The *Objecteering/ClearCase* module does not handle the following configuration management functions:

- ◆ the administration of ClearCase VOB
- ◆ the merge of Objecteering/UML objects
- ◆ the manual selection of earlier versions of the configuration's objects

These functions are handled through the standard ClearCase interface.

Consistency rules

ClearCase commands can only be run if:

- ◆ a view has been defined.
- ◆ the configuration parameters of the module have been correctly entered (positioning of ClearCase and positioning of the VOB).
- ◆ the ClearCase environment is accessible.

Functions

Using the *Objecteering/ClearCase* module, it is possible to:

- ◆ Check-in and/or check-out a model element
- ◆ Undo check-out
- ◆ Update the Objecteering/UML model in relation to the ClearCase view
- ◆ Apply a label
- ◆ Open a history browser
- ◆ Open a version tree browser


In Unix, a view must have been defined and positioned (using the "*cleartool setview <view-tag>*" command).

Glossary

- ◆ *View*: A view which you use to select a particular version of files placed in configuration.
 - ◆ *VOB*: Versioned Object Base. A VOB stores all versions of all objects.
-

Using the Objecteering/ClearCase module

Selecting the Objecteering/ClearCase module in your UML modeling project

Launch the *Objecteering/UML Modeler* tool on your UML modeling project. The  "UML modeling project modules" icon launches the window used to select the module (as shown in Figure 7-1).

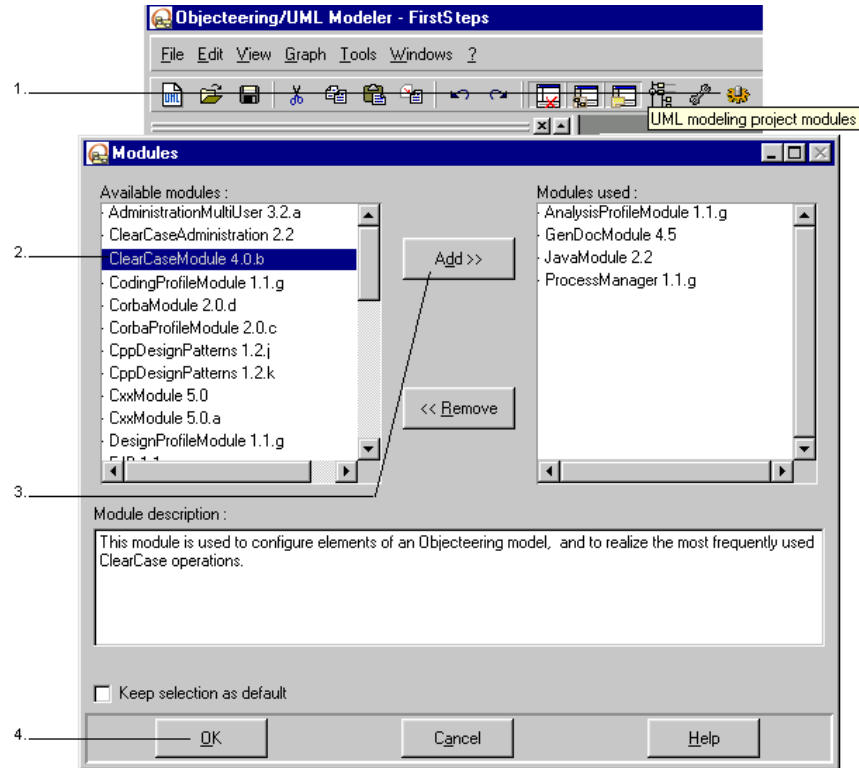



Figure 7-1. Selecting the *Objecteering/ClearCase* module

Chapter 7: The Objecteering/ClearCase module

Steps:

- 1 - Click on the  "UML modeling project modules" icon.
- 2 - Select the *Objecteering/ClearCase* module from the available modules list on the left-hand side of the window.
- 3 - Click on the "Add" button. The *Objecteering/ClearCase* module then appears in the right-hand "Modules used" column.
- 4 - Click on "OK" to confirm.

Two case scenarios are possible:

- ◆ A new user wishing to connect to a view. In this case, module configuration parameters should simply be updated. By selecting the module, a wizard is used to initialize these parameters.
- ◆ The project is new and has not yet been archived in a VOB. This work is particularly destined for the build manager, who must export the Objecteering/UML model in the form of a file and migrate these files into a VOB. This case scenario does not allow the module to be selected, but can be used to create the view, through a wizard.

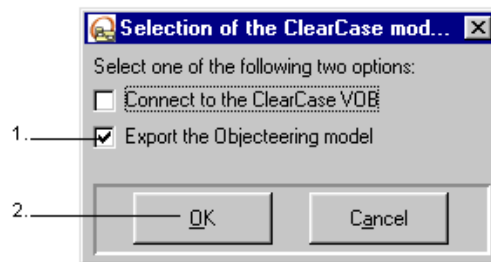


Figure 7-2. Choices available when selecting the module

Steps:

- 1 - Select the desired options.
- 2 - Confirm.

Exporting the Objecteering/UML model

To be put in version in a VOB, a model must be generated in the form of files. The "*Export the Objecteering/UML model*" option is used to generate all files which are indispensable between Objecteering/UML and ClearCase.

This option is only used once, by the administrator, to create the ClearCase VOB. The ClearCase migration directory contains files representing the model which are archived in ClearCase. It contains three types of import (complete, interface, structural) for the same element in a single file.

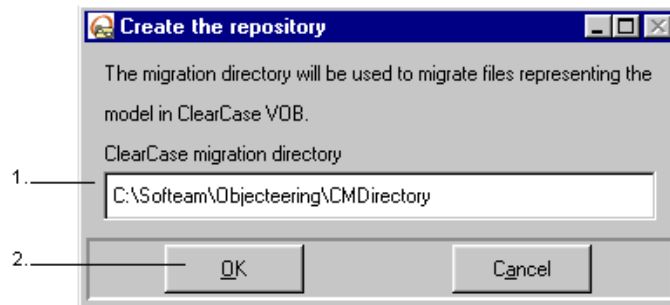


Figure 7-3. Creating files which represent the model

Steps:

- 1 - In the "*ClearCase migration directory*" field, define the migration directory. The default value of this field is <OBJING_PATH>\CMDirectory.
- 2 - Confirm.

Note: The directory entered in this dialog box is not the definitive directory. The administrator can change these values at any time after the migration of the files into ClearCase or at any other time during the development cycle.

Chapter 7: The Objecteering/ClearCase module

Once the model has been exported into the directory entered, the administrator is informed of the actions to be carried out.

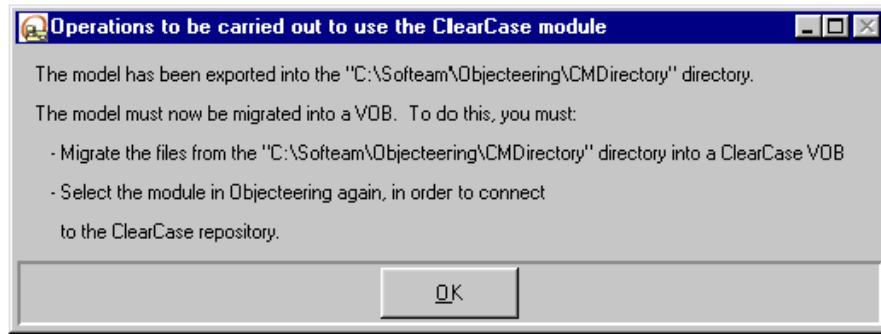


Figure 7-4. Actions to be carried out after creating the repository

This operation cannot be used to select the module, but only to generate files which can be migrated into ClearCase.

The following steps are used to update the ClearCase environment, in order to integrate the Objecteering/UML model into this environment. These steps must be carried out in the ClearCase environment.

Configuring the ClearCase environment

The *Objecteering/ClearCase* module cannot be used to create the repository directly within ClearCase. This must be carried out manually. Once the module has externalized the directory for ClearCase, the directory must be injected into ClearCase. This operation is carried out over two stages:

- 1 - The creation of an archive specific to ClearCase, which contains all the directories and all the externalized files. This operation is carried out via the "clearexport_ffile" command. We recommend that you rename the CMDirectory directory, using the name of the project, in order to better distinguish projects in the VOB.
- 2 - The import of this specific archive into ClearCase. This operation is carried out via the "clearimport" command. In Windows, it is imperative that you launch this ClearCase command with the "pcase" option, so as to retain capitals present in the names of directories.

The ClearCase environment is now ready.

Note: For further information on ClearCase functions, please refer to the ClearCase user guides.

Connecting to the ClearCase VOB

All the environments have now been defined. All that remains is to select the *Objectteering/ClearCase* module and to position it on the ClearCase VOB.

To initialize your Objectteering/UML modeling project with the contents of the repository, select the *Objectteering/ClearCase* module in your UML modeling project and choose the "Connect to the ClearCase VOB" option.

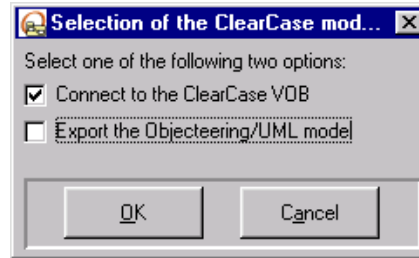


Figure 7-5. Connecting to the ClearCase VOB

Steps:

- 1 - Enter the shared project directory in VOB.
- 2 - Confirm.

After these operations, the project contained in the previously specified view is imported.

The Objectteering/UML project is now initialized with the name of the view and the positioning of the VOB. It will no longer be possible to use the project with a view other than that used to initialize the project.

To modify the link between the view and the Objectteering/UML project, the "Modify the parameters" administration module command should be used.

Objecteering/ClearCase commands

Overview

The following commands are specific to the *Objecteering/ClearCase* module.

| The ... command | is used to ... |
|-----------------|---|
| Apply label | attach a label to a unit version defined by a view. |
| History | open the History window of the current element. |
| Version tree | open the version tree window for the current element. |
| Properties | display an element's properties. |

The "History" command

This command starts a history browser on the current element, to display event records.

For further information on this dialog box, please refer to the related ClearCase documentation.

The "Version tree" command

This command starts a version tree browser on the current element.

For further information on this dialog box, please refer to the related ClearCase documentation.

The "Apply label" command

This command allows you to attach a label to a modeling unit and any components it may have.

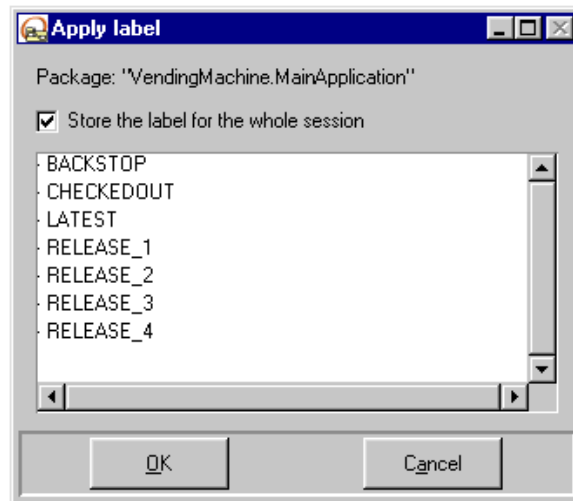


Figure 7-6. Applying a label

Note: This command cannot be used to create new labels. The creation of new labels is carried out from the ClearCase interface.

The "Store the label for the whole session" tickbox allows you not to enter a label for all components. When labeling is carried out in depth, the label entered the time before is then re-used.

The "Properties" command

The "Properties" command is used to display an element's properties.

Effects of commands in ClearCase

Introduction

This section describes the ClearCase commands run when *Objecteering/ClearCase* module commands are executed.

The following table details the ClearCase commands which are run for each Objecteering/UML command. Users must have the right to run these commands, as well as read-write rights for all the files which make up the repository.

The Objecteering/UML interface suggests that a comment be entered when an element is created or checked-in. Certain comments are added automatically by the *Objecteering/ClearCase* module, for example, during the check-out/check-in of a directory for the creation of a new element.

Note: So as to avoid complicating the table, it should be pointed out that certain operations are carried out with comments and others without comments. In the description of the commands, we do not, therefore, indicate options concerning comments.

| Objecteering/UML commands | ClearCase commands |
|---------------------------|---|
| Check-out | cleartool checkout <file> |
| Check-in | <p>If the element has not been modified, the Objecteering/UML "<i>Check-in</i>" command runs the ClearCase "<i>Undo check-out</i>" command. This operation is used to limit the number of versions of an element in ClearCase.</p> <p>To know whether or not the element has been modified before running the check-in:</p> <pre>cleartool diff -predecessor <file></pre> <p>Check-in of the element:</p> <pre>cleartool checkin <file></pre> |
| Check-in and check-out | Check-in followed by a check-out |

| Objecteering/UML commands | ClearCase commands |
|--|--|
| Create a model element (NameSpace except Enumeration) | Check-out of the directory: cleartool checkout <directory> Creation of the element: cleartool mkelem -ci -ptime <file> Check-in of the directory: cleartool checkin <directory> |
| Destroy a model element (NameSpace except Enumeration) | Check-out of the directory: cleartool checkout <directory> Destruction of the element: cleartool rmname <file> Check-in of the directory: cleartool checkin <directory> |
| Undo check-out | cleartool uncheckout -rm <file> |
| Apply label | Retrieval of the different labels: cleartool lstype -kind ldtype -fmt "%En " -invob <VOB> Attachment of a label: cleartool mlabel <label> <file> |
| History | cleartool lshistory -graphical <file> |
| Version tree | cleartool lsvtree -graphical <file> |
| Import element from repository | No ClearCase operations |
| Project check-in | Check-in of all the elements of the Objecteering/UML project in check-out |
| Properties | cleartool describe -graphical <file> |

Chapter 8: The Objecteering/SCC
module

Introduction to the Objecteering/SCC module

Overview

The *Objecteering/SCC* module is based on an SCC API. This SCC API is used to interface development tools with different CMS (configuration management system) tools which use it (SCC providers). CMS tools which implement this SCC API include PVCS, VSS, ClearCase, CVS and CM Synergy. The SCC API only exists in the Microsoft Windows environment.

The *Objecteering/SCC* user guide concentrates particularly on the PVCS provider.

The *Objecteering/SCC* module does not substitute the provider. The administration of this tool is not, therefore, available from *Objecteering/UML*.

In order to use the *Objecteering/SCC* module, the *Objecteering/UML* tool must already have been installed. The SCC provider used must also have been installed. As far as PVCS is concerned, you should first install PVCS Version Manager and then SCC.

The *Objecteering/SCC* module can only be deployed on a Windows platform.

Limitations

In order to take into account possible new elements created by other users, the "*Folder/Update Project Folder*" menu command in PVCS Version Manager V6.0 must be run.

Please note that with PVCS, the name used is that given by PVCS Version Manager V6.0 in the "*Configure Project/Network*" menu. By default, this is positioned to *HOST*, in other words, the current user name is the Windows login name.

Functions

Using the *Objecteering/SCC* module, it is possible to:

- ◆ Check-in and/or check-out a model element
 - ◆ Undo check-out
 - ◆ Update the Objecteering model in relation to the SCC provider
 - ◆ Open a history browser
 - ◆ Obtain information on an element by opening the properties dialog box
 - ◆ Display the provider options dialog box (according to the provider in question, it is possible to attach a label via this dialog box)
-

Using the Objecteering/SCC module

Presentation

During module selection and unselection operations, consistency must be maintained between the information contained in the Objecteering/UML model and in that of PVCS. The *Objecteering/SCC* module manages these operations, in order to guarantee this consistency.

Selecting the module

After the *Objecteering/SCC* module has been selected for the new UML modeling project, a series of operations is automatically carried out, in order to ensure consistency between the information in the CMS and that contained in the Objecteering/UML model.

First, a dialog box (shown in Figure 8-1) opens, asking the user to enter details on the exchange directory.

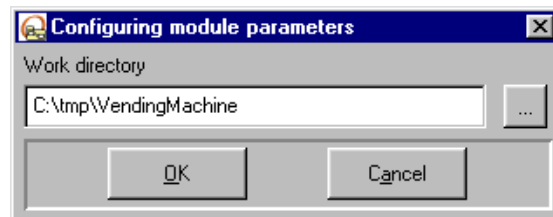


Figure 8-1. Entering the exchange directory

The module then attempts to open the PVCS project. If the PVCS project does not exist, a dialog box appears, through which a new project may be created (Figure 8-2).

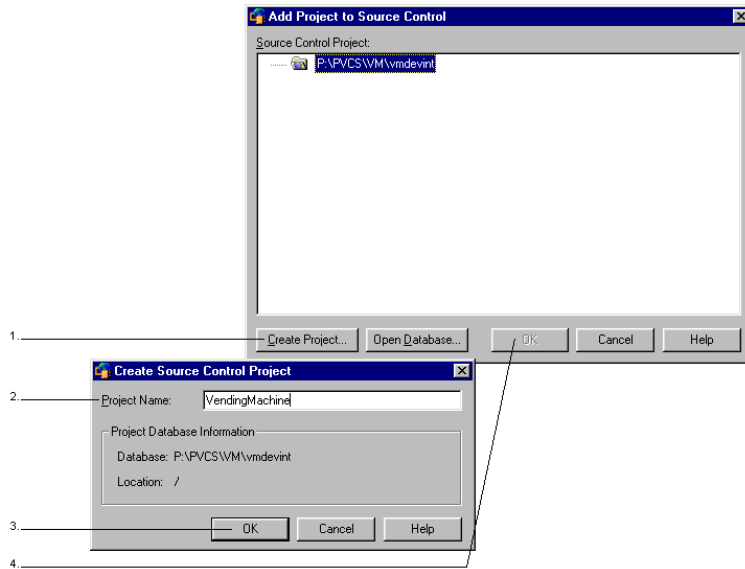


Figure 8-2. Creating a new PVCS project

Steps:

- 1 - Click on the "Create Project..." button. The "Create Source Control Project" window then appears, in which the name of the project should be entered.
- 2 - A project name is proposed. If this name is different to that entered in Objecteering/UML, change it.
- 3 - Click on "OK" to confirm.
- 4 - Click on "OK" to confirm.

Important: The project name must be identical to that entered in Objecteering/UML!

After this, all model elements are imported into Objectteering/UML, and are put into read-only mode.

If the PVCS project exists, the window shown in Figure 8-3 appears, through which the project may be selected.

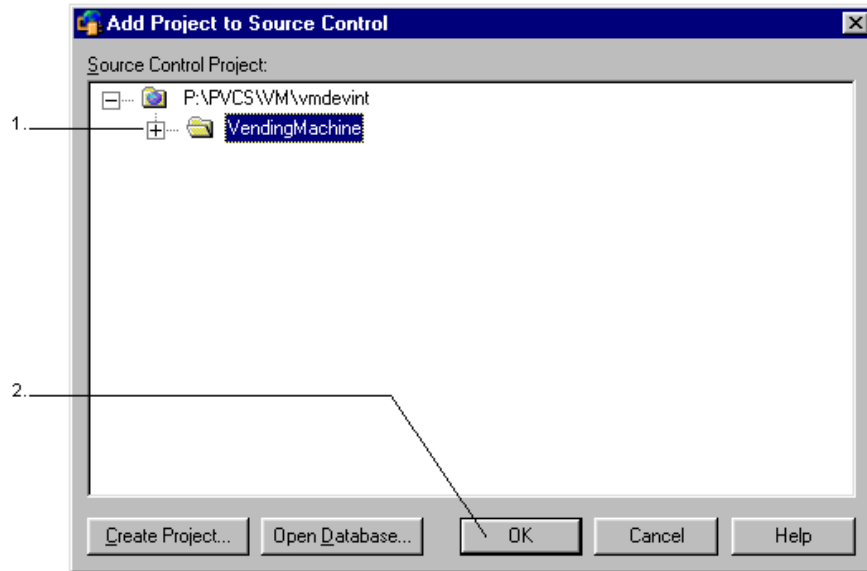


Figure 8-3. Selecting a PVCS project

Steps:

- 1 - Select the PVCS project you wish to use.
- 2 - Confirm by clicking on "OK". The structure of the project is then imported into Objectteering/UML.

Chapter 8: The Objecteering/SCC module

Elements are then presented in Objecteering/UML with a bitmap symbol, indicating that they are locked and may not be modified (Figure 8-4).

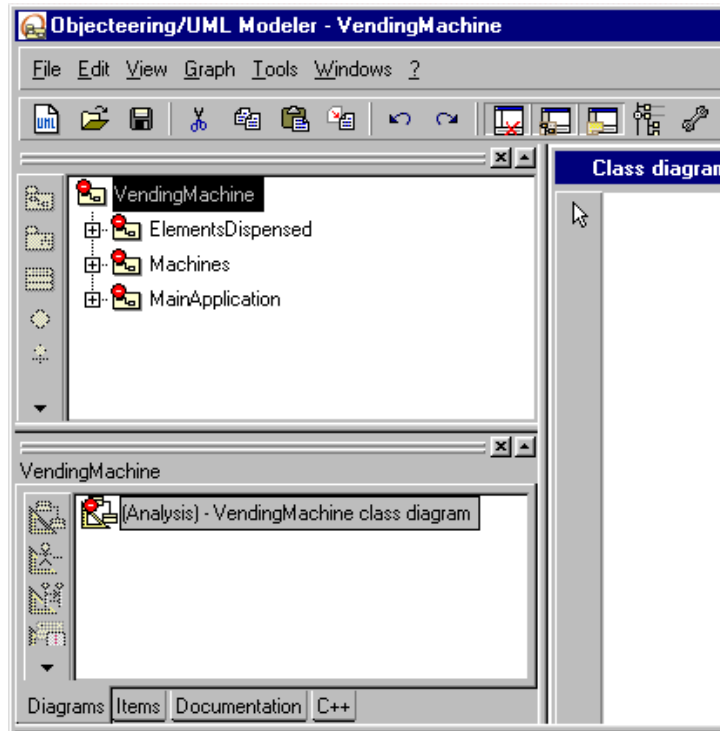


Figure 8-4. Representation of model elements in Objecteering/UML

The information contained in the Objecteering/UML modeling project is consistent with that of the PVCS project. All commands in the SCC menu are now available.

Objectteering/SCC commands

Overview

The following commands are specific to the *Objectteering/SCC* module.

| The ... command | is used to ... |
|-----------------|--|
| History | open the History window of the current element. |
| Properties | display the element's properties. |
| Options | open the provider's options dialog box. For PVCS, this dialog box is used to launch PVCS or to attach a label to the files corresponding to the Objectteering/UML model. |

History

The "*History*" command is used to obtain details on the modifications made to an element.

It should be noted that the file which manages information relative to model elements has a name which is internal to Objecteering/UML (principally constructed from element identifiers).

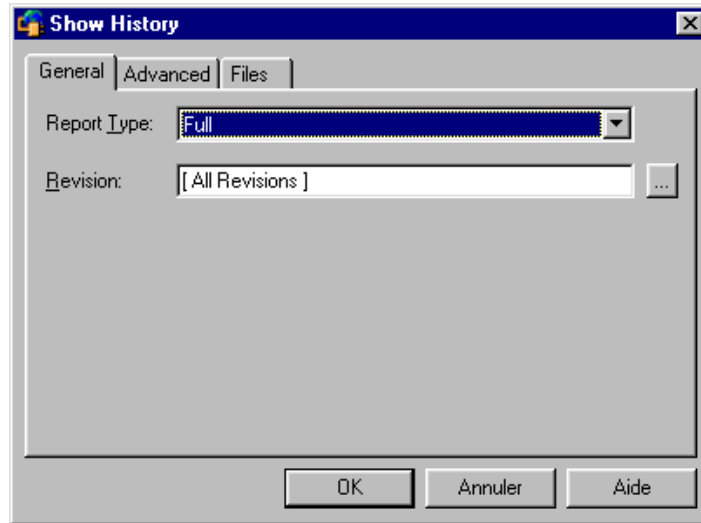


Figure 8-5. An element's history

For further information on this dialog box, please refer to the related PVCS documentation.

Properties

The "*Properties*" command is used to obtain information on the state of the file in relation to the model element on which the command has been run.

It should be noted that the file which manages information relative to model elements has a name which is internal to Objecteering/UML (principally constructed from element identifiers).

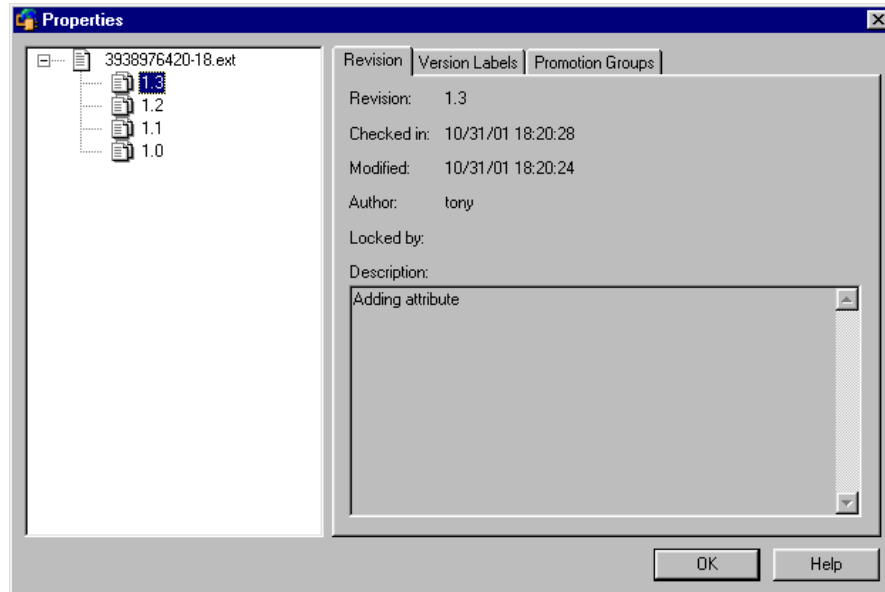


Figure 8-6. Information relative to an element

For further information on this dialog box, please refer to the related PVCS documentation.

Options

The "Options" command is used to display the provider options, allowing the user to define preferences regarding provider functioning. For example, PVCS proposes:

- ◆ the attaching, renaming and deleting of a label on the files of a PVCS project
- ◆ the definition of the default database
- ◆ the definition of options on the check-in or the check-out

This command is only available on packages, and can be called from any package.

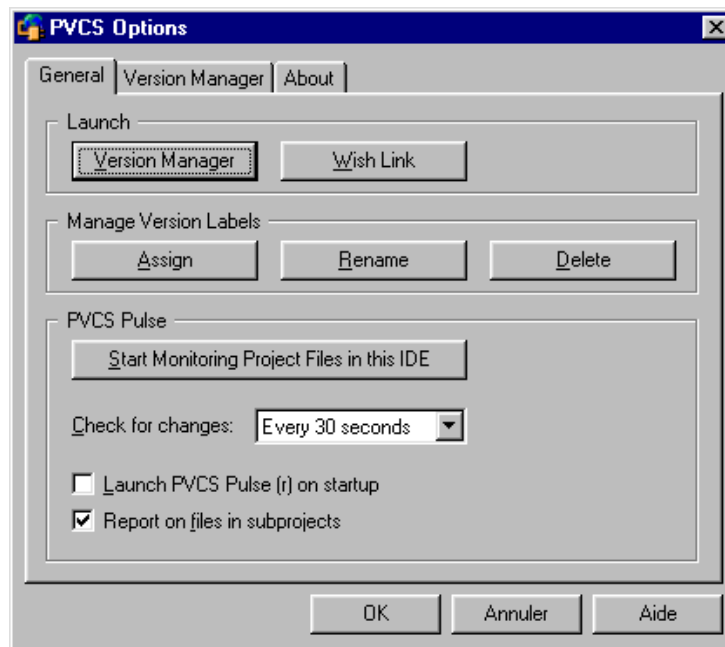


Figure 8-7. Provider options

For further information, please refer to the related PVCS documentation.

Assigning a label

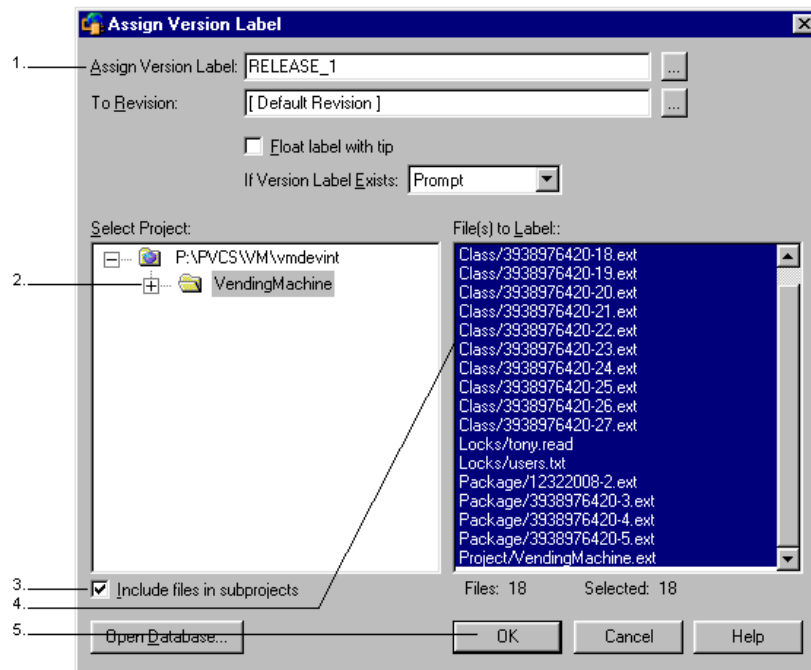


Figure 8-8. Assigning a label

Steps:

- 1 - After clicking on the "Assign" button in the window shown in Figure 8-7, enter the name of the label in the "Assign version label" field.
- 2 - Select the project.
- 3 - Check the "Include files in subprojects" tickbox.
- 4 - Select all the files.
- 5 - Click on "OK" to confirm.

Advanced options

When the "Advanced options" module parameter has been activated, additional options are available on the "Check-out" and "Import..." commands. These options are used to retrieve an earlier version of the model or a model element.

When the model element retrieval mode is hierarchical, a warning window is displayed (as shown in Figure 8-9), to indicate the importance of retrieving elements according to a label. In this case, all model elements are retrieved according to the label in question, which means that all elements must have this label. If this is not the case, certain elements cannot be retrieved and the imported model runs the risk of being inconsistent.

Similarly, if a version number is entered rather than a label for a hierarchical retrieval, the number may not exist for certain elements. In this case, some elements will not be retrieved properly, and the model runs the risk of being inconsistent.

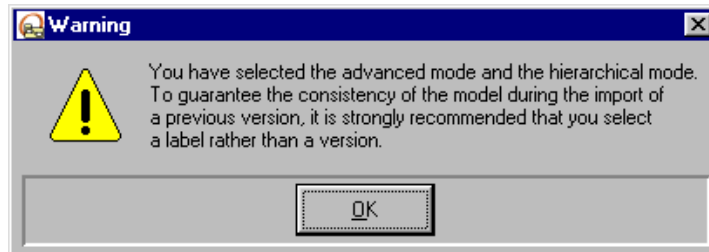


Figure 8-9. Warning window which appears during a hierarchical retrieval

After this warning window, the PVCS provider options window opens, in which the revision which is to be retrieved can be entered (Figure 8-10).

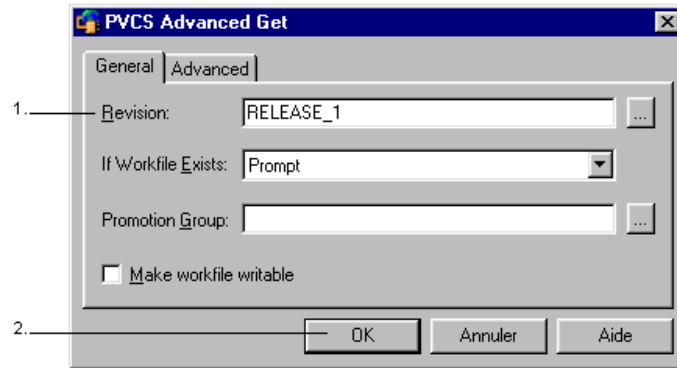


Figure 8-10. Advanced mode for "Import..." commands

Steps:

- 1 - Select a label or a version in the "Revision" field.
- 2 - Click on "OK" to confirm.

Retrieval by label in hierarchical mode is used for functional retrieval. In this way, a set of model elements which are consistent between themselves can be transferred into Objecteering/UML.

In non-hierarchical mode, a version or a label can be specified. In this case, the user wishes to retrieve an earlier version of a particular element.

In the "Advanced options" mode, element retrieval only checks the read-only state. If the retrieval of an element requires the modification of an element in read-only mode, the retrieval operation is aborted. The user must then manually resolve the problem, by importing or checking-out the elements which are modified by the element retrieval operation. This case arises when element retrieval requires the retrieval of other dependent elements which are components of the elements in read-only.

Objectteering/SCC module parameters

The *Objectteering/SCC* module provides three additional module parameters in the "General settings" parameter set.

| The ... parameter | is used to ... |
|--|---|
| Work directory | <p>define the directory for the exchange of information between Objectteering and the SCC provider. This temporary directory parameter is initialized when the module is selected.</p> <p>When this parameter is modified, the following steps should be carried out in order to take into account the modifications:</p> <ol style="list-style-type: none"> 1) Restart Objectteering/UML. 2) Check the "Open the SCC project for every operation" tickbox. |
| Advanced options | <p>refine the execution of SCC commands specifically for the "Check-out" and "Import from repository" commands. For example, it is possible to retrieve an earlier version of an element, by providing a specific version number or a label name. For this, the option must be activated before execution of the "Import element from repository" or "Check-out" command.</p> |
| Open the SCC project for every operation | <p>is used to close and open the SCC project before every command. This parameter is very useful if parameters have been modified in the SCC provider and reloading of these SCC parameters has to be forced.</p> |

Index

- .cout 5-16
- .cout files 6-3
- .lock files 6-3
- .ofp files 1-3, 1-12
- .read 5-16
- .read files 6-3
- .write 5-16
- .write files 6-3
- <username>.read file 5-16
- Activating consistency checks 3-11
- Actor 1-18
- Actors 1-9
- Adding and modifying diagrams 3-7
- Adding comments 7-13
- Adding elements 3-6
- Adding links between elements 3-8
- Adding non-oriented links 3-8
- Adding oriented links 3-9
- Administration module 1-14, 5-9
- Administration module commands 5-9
 - Generate element in repository 5-10
 - Modify the parameters 5-10, 7-10
 - Synchronize element state in project 5-10
 - Unlock repository 5-10
- Adminstrating ClearCase VOB 7-3
- Advanced options 8-14, 8-16
- Aggregations 3-8
- API 1-18
- Apply label 7-11
- Applying a label 7-4, 7-12
- ASCII files 1-9, 6-6
- Assigning a label 8-13
- Associations 3-8
- Attaching a label 7-12, 8-4, 8-9, 8-12
- Build manager 7-6
- C++ work products 3-13
- Changing bitmaps 5-7
- Check-in 1-9, 1-14, 1-15, 1-18, 2-3, 2-4, 2-5, 2-10, 3-5, 3-6, 3-10, 3-11, 3-12, 4-3, 5-5, 5-9, 5-13, 5-16, 6-3, 7-4, 8-4
- Check-in and check-out 2-3, 2-5, 2-12
- Check-out 1-9, 1-14, 1-15, 1-16, 1-18, 2-3, 2-4, 2-8, 3-5, 3-6, 3-8, 3-9, 3-11, 3-13, 4-3, 4-7, 5-5, 5-9, 5-13, 6-3, 7-4, 8-4
- Check-out: 2-5
- Circular dependencies 3-10
- Circular renaming 3-7
- Class 1-9, 1-18
- Classes 1-9, 3-8
- ClearCase 8-3
- ClearCase 4.0 7-3
- ClearCase 4.1 7-3
- ClearCase 4.2 7-3
- ClearCase 5.0 7-3
- ClearCase commands 7-13
 - Apply label 7-4, 7-11
 - cleartool checkin <directory> 7-14
 - cleartool checkin <file> 7-13
 - cleartool checkout <directory> 7-14
 - cleartool checkout<file> 7-13
 - cleartool describe -graphical <file> 7-14
 - cleartool diff -predecessor <file> 7-13
 - cleartool lshistory -graphical <file> 7-14

cleartool lstype -kind ldtype -fmt 7-14
cleartool lsvtree -graphical <file> 7-14
cleartool mkelem -ci -ptime <file> 7-14
cleartool mklabel <label> <file> 7-14
cleartool rmname <file> 7-14
cleartool uncheckout -rm <file> 7-14
History 7-11
History browser 7-4
Properties 7-11
Version tree 7-11
Version tree browser 7-4
ClearCase interface 7-3
ClearCase migration directory 7-7
ClearCase operations 1-4, 7-3
clearexport_file command 7-9
clearimport command 7-9
cleartool checkin <directory> 7-14
cleartool checkin <file> 7-13
cleartool checkout <directory> 7-14
cleartool checkout <file> 7-13
cleartool describe -graphical <file> 7-14
cleartool diff -predecessor <file> 7-13
cleartool lshistory -graphical <file> 7-14
cleartool lstype -kind ldtype -fmt 7-14
cleartool lsvtree -graphical <file> 7-14
cleartool mkelem -ci -ptime <file> 7-14
cleartool mklabel <label> <file> 7-14
cleartool rmname <file> 7-14
cleartool setview <view-tag> 7-4
cleartool uncheckout -rm <file> 7-14
CM Synergy 8-3
CMS 1-18, 8-5
CMS tools 8-3
Command line 1-17
Commands available on all elements 2-3
Commands available on elements in read-only mode 2-3, 2-7
Commands available on elements in read-write mode 2-3
Comments 7-13
Communication links 3-8
Component 1-18
Component multi-user atomic unit file 1-18
Components 3-8
Configuration management 1-9, 7-3
Configuration management system 1-18, 8-3
Configuring the ClearCase environment 7-9
Configuring the Objectteering/UML teamwork modules 5-3
Connecting to the ClearCase VOB 7-10
Consistency checks 1-16, 3-11
Consistency rules 7-3
Console 5-15, 6-10
Copy/paste operation 3-7
Create source control project 8-6
Creating a model database 3-5
Creating a new PVCS project 8-6
Creating an initial model 3-3
Creating new labels 7-12
Cut/paste operation 3-7
CVS 8-3

- Data type 1-18
- Database 1-6
- Deactivating consistency checks 3-11
- Default database 8-12
- Defining a default database 8-12
- Defining options 8-12
- Defining principal packages 3-4
- Defining the initial model 3-4
- Delete command 3-6
- Deleted directory 1-10
- Deleting a label 8-12
- Deleting elements 3-6
- Deleting links between elements 3-9
- Deleting locks 5-10
- Dependencies 3-4
- Dependency cycles 3-11
- Destroying referenced elements 3-6
- Developing and merging
 - Objecteering/UML objects 7-3
- Development space 1-3, 1-5
- Diagrams 3-7
- Directories 1-8
- Directory hierarchy 6-6
- Element history 8-10
- Element identifiers 1-10, 4-4, 8-11
- Element versions 6-11
- Embedding elements 3-6
- Embedding units 2-3
- Entering the exchange directory 8-5
- Enterprise Edition 1-4, 1-6
- Example of an import 4-8
- Exchange directory 8-6
- Exchanging externalized data 1-5
- Exchanging model information 1-3
- Exchanging models 1-6
- Explorer 3-6, 4-5
- Exporting the Objecteering/UML model 7-7
- External edition 3-12
- Externalization 5-11, 7-9
- Externalization binary
- Externalization command 5-11
- Externalization function 1-9
- Externalization hierarchy 1-11
- Files 1-8
- First level package 3-7
- General settings parameter set 5-4
- Generalizations 3-9
- Generate element in repository 5-11
- Generating ASCII files 1-9
- Generating C++ sources 3-13
- Generating Java sources 3-13
- Generation work products 5-6
- Get information 6-10
- Glossary 7-4
- Graphic representation of the read-only mode 5-7
- Graphic representation of the read-write mode 5-7
- Group work 1-18
- Hierarchical mode 2-4, 2-8, 4-3, 5-5, 5-11, 6-10, 8-14
- Hierarchical retrieval 8-14
- History 7-11, 8-9
- History browser 7-11
- Identifiers 4-4
- Identifying model elements 4-4
- Implementation links 3-9
- Import 1-9
- Import commands 3-5
- Import complete element from repository 3-5

- Import dialog box 4-5
- Import element from repository* 2-3
- Import element using current mode 3-7
- Import from repository 1-12, 1-16, 2-4, 2-5, 2-8, 3-13, 4-3, 5-5, 5-9, 5-13
- Import logic 4-4
- Import service 1-6
- Imported objects 4-7
- Importing a whole UML modeling project 4-9
- Importing elements 4-3, 4-5
- Importing elements between UML modeling projects 1-5
- Importing elements from other databases 4-3
- Initial model 3-4
- Initial model structure 3-4, 6-4
- Initializing private work spaces 6-7
- Initializing the repository 6-4
- Integration space 1-3
- Interfaces 3-9
- Internalization 1-16
- Internalization function 1-9
- Internalization/externalization mechanism 1-9
- Inter-project import 1-18
- Java work products 3-13
- Label 1-18
- Labels 8-14
- Link circularity problems 3-10
- Links 3-3, 3-8, 4-4
- Location of Objectteering/UML modeling projects 1-10
- Locking checks 1-16
- Locking files 1-10
- Locking mechanism 1-18
- Locks 3-11
- Locks directory 1-10, 5-16
- Managing generated files 3-3
- Managing work products 3-3
- Managing work spaces 1-3
- Manually selecting earlier versions of configuration objects 7-3
- Microsoft SCC 1-4
- Microsoft SCC API 1-4
- Migrating Objectteering/UML 1-12
- Migration directory 7-7
- Migration from an earlier version of Objectteering/UML 1-12
- Migration from Objectteering/UML 5.2.1 1-12
- Model elements 1-18, 2-7, 3-7, 3-8
 - Actors 3-8
 - Dataflows 3-8
 - Use cases 3-8
- Model exchange facility 1-6
- Model root name 6-4, 6-7
- Model structure 3-4
- Modify module parameter configuration 5-3
- Modify the parameters 5-13
- Modifying code 3-12
- Modifying configuration 5-3
- Modifying module parameters 5-10
- Modifying notes 3-12
- Module behavior 3-6
- Module configuration 5-9
- Module parameters 1-13, 5-3
 - Confirm operations 2-4, 2-11, 2-13, 5-5, 5-11
 - Default values 5-3
 - Hierarchical operations 5-5, 5-9
- Log 5-5
- Memorize actions in the log 5-5

- Update generation work products after import 5-6
- Update generation work products before check-in 5-6
- Update project at start-up 5-5
- Work directory 5-12
- Module restrictions
 - Destroying a referenced element 3-6
- Module selection 8-5
- Module unselection 8-5
- Modules used 1-17
- Moving elements 3-7
- Multi-user atomic unit 1-18
- Multi-user atomic units 1-10, 2-3, 3-3, 5-10
- Multi-user commands
 - Check-in 3-6
 - Get information 6-10
 - Properties 6-10
- Multi-user directory 3-4
- Multi-user projects 1-5
- Multi-user space 1-8, 1-16
- Multi-user work area 1-18
- Multi-user work space 1-9
- multiuser.lock file 5-16
- Node 1-18
- Nodes 3-8
- Non-hierarchical mode 2-4, 4-3, 4-7, 5-5
- Non-oriented associations 4-7
- Non-oriented links 3-8
- obj_extreport 1-10
- Objecteering
 - Work spaces 1-5
- Objecteering/Administrating
 - Objecteering Sites 1-3, 1-17
- Objecteering/ClearCase 1-4
- Objecteering/ClearCase module functions
 - Apply label 7-12
- Objecteering/Introduction 1-3, 1-13
- Objecteering/Multi-user 1-4
- Objecteering/SCC 1-4
- Objecteering/UML crashes 3-11
- Objecteering/UML database 1-9
- Objecteering/UML environment 1-3
- Objecteering/UML first steps 1-3
- Objecteering/UML model 1-4
- Objecteering/UML Modeler 1-3
- Objecteering/UML modeling project 1-9, 1-18
- Objecteering/UML objects 7-3
- Objecteering/UML versions 1-15
- Objecteering/XMI 1-6
- Objects 3-8
- objing 1-17
- Open a history browser 7-4
- Open a version tree browser 7-4
- Open the SCC project for every operation 8-16
- Opening a history browser 8-4
- Optimization mode 5-13
- Optimization phases 1-10
- Options 8-9
- Organization
 - Directories 1-10
 - Principle 1-9
- Oriented links 3-8
- Overview of teamwork in Objecteering/UML 1-3
- Package 1-18
- Packages 1-9
- Personal Edition 1-6

- Physical organization 1-8
- Principal packages 3-4
- Professional Edition 1-6
- Project check-in 1-12, 2-3, 2-5, 2-13, 5-13
- Properties 2-5, 6-11, 7-11, 8-4, 8-9
- Provider functioning preferences 8-12
- Provider options 8-4
- PVCS 8-3, 8-5
- PVCS project 8-6
- PVCS provider options window 8-15
- PVCS Version Manager 8-3
- Rational ClearCase 1-4
- Read-only mode 1-18, 2-3, 2-5, 2-7, 5-7, 5-9, 8-7, 8-15
- Read-write mode 1-19, 2-3, 2-5, 2-9, 3-4, 5-7, 5-9
- Reference links 3-10, 4-7
- Referenced element 3-6
- References 3-6
- Referencing links 3-11
- Removable consistency checks 1-16
- Renaming a label 8-12
- Renaming elements 3-7
- Repository 1-19
- Repository location 6-11
- Repository locks 5-16
- Rereading ASCII files 1-9
- Reserved elements 3-7
- Reserved units 2-8
- Reserving units 2-4
- Restoring 6-3
- root directory 1-10
- Saving 6-3
- Saving elements 3-6
- SCC 1-4, 1-19, 8-3
- SCC API 8-3
- SCC commands 8-9
 - History 8-4, 8-10
 - Options 8-4, 8-12
 - Properties 8-4, 8-11
- SCC module parameters
 - Advanced options 8-16
 - Open the SCC project for every operation 8-16
 - Work directory 8-16
- SCC provider 8-3
- SCC provider repository 1-19
- Selecting a module 6-5
- Selecting the
 - Objecteering/ClearCase module 7-5
- Selecting the SCC module 8-5
- Several developers working together at the same time 1-3
- Shared elements 1-18
- Signal 1-18
- Site 1-5
- Source Code Control 1-19
- Stopping commands 1-17
- Summary of commands available in read-only mode 2-5
- Summary of commands available in read-write mode 2-5
- Summary of commands available on all elements 2-5
- Supporting teamwork operations 1-4
- Synchronization 3-11, 5-10
- Synchronize element state in project 5-15
- Teamwork commands 2-3
 - Check-in 1-14, 2-4, 2-10, 3-6, 3-10, 3-11, 3-12, 4-3, 5-5, 5-9, 5-13, 5-16, 6-3, 7-4, 8-4

- Check-in and check-out 2-5, 2-12
- Check-out 1-14, 1-16, 1-18, 2-4, 2-8, 3-6, 3-8, 3-11, 3-13, 4-3, 4-7, 5-5, 5-9, 5-13, 6-3, 7-4, 8-4
- Import from repository 1-12, 1-16, 2-4, 2-8, 3-13, 4-3, 5-5, 5-9, 5-13
- Project check-in 1-12, 2-5, 2-13, 5-13
- Properties 2-5, 2-13
- Undo check-out 2-5, 2-12, 3-6, 3-11, 5-16, 6-3, 7-4, 8-4
- Teamwork module 1-18
- Teamwork modules 1-4
- Teamwork operations 1-4
- Teamwork repository 1-12
- tmp directory 1-10
- Tools/Import 4-9
- Transfer function 4-3
- Transfer logic 4-4
- Transferring elements 4-3
- UML model root 4-3, 4-7, 6-7
- UML modeling project 1-3
- UML modeling project data 1-3
- UML modeling projects 1-5
- UML models 3-4
- Undo check-out 1-15, 2-3, 2-5, 2-12, 3-6, 3-11, 5-16, 6-3, 7-4, 8-4
- Undo/redo operations 1-15
- Universal model identification mechanism 1-6
- Unlock repository 5-16
- Unselecting the module 2-6
- Updating generation work products after import 3-13
- Updating generation work products before check-in 3-12
- Updating the Objectteering/UML model 6-3, 7-4, 8-4
- Updating your model 1-9
- Use case 1-18
- Use cases 1-9
- Use links 3-11
- User 1-19
- users.txt file 5-16
- Version control systems 1-4
- Version label 8-13
- Version numbers 8-14
- Version tree 7-11
- Version tree browser 7-11
- Versioned Object Base 7-4
- Views 7-3, 7-6
- VOB 7-3, 7-4, 7-6
- VSS 8-3
- Work directory 8-16
- Work product options parameter set 5-6
- Work products 3-3
- Work space 1-3
- XMI 1-6
- XMI exchange service 1-6